VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Implementace objektově-relačního mapování v datové vrstvě informačního systému pro platformu Java

Implementation of Object-Relational Mapping in Data Layer of an Information System for the Java Platform

2010 Lubomír Turčík

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2010 .....

V úvodu bych chtěl poděkovat panu doc. Ing. Michalu Krátkému, Ph.D. za ochotnou pomoc, kvalitní odborné vedení a přínosné pokyny při vypracovávání mé diplomové práce.

Abstrakt

Práce se zabývá problematikou objektově-relačního mapování v datové vrstvě informačního systému. Ukazuje možnosti zajištění perzistence dat, probírá techniky používané při ORM a představuje standardy v této oblasti. Seznamuje také čtenáře s historií SŘBD a představuje možnosti jejich využití. Prezentuje vybrané ORM frameworky a ukazuje možnosti jejich využití při tvorbě informačních systému. Cílem této práce je návrh a implementace ORM frameworku pro platformu Java. Tento framework je porovnán s vybranými ORM Frameworky. V závěru je provedená analýza porovnávání a jsou definovány výhody a omezení implementovaného frameworku.

Klíčová slova: informační systém, datová vrstva, objektově-relační, mapování, Java, perzistence, SŘBD, Hibernate, EclipseLink

Abstract

The thesis focuses on the issue of Object - Relational Mapping in the Data Layer of Information Systems. It demonstrates different possibilities of ensuring data persistence, examines techniques used with ORM and introduces standards followed in this field. Moreover, the reader gets acquainted with the history of the databases and with the area of their possible use. The work presents selected ORM Frameworks and illustrates different possibilities of their use in the process of creating Information System .The aim of this thesis is to propose and implement ORM Framework for Java Platform. This framework is compared to other selected ORM Frameworks. In conclusion, the comparison of frameworks is analysed and the advantages and limitations of the implemented framework are defined.

Keywords: information system, data layer, Object-relational, mapping, Java, persistence, DBMS, Hibernate, EclipseLink

Seznam použitých zkratek a symbolů

DDL	–	Data Definition Language
HTML	–	HyperText Markup Language
J2SE	–	Java 2 Standart Edition
J2EE	–	Java 2 Enterprise Edition
JDBC	–	Java Database Connectivity
JDO	–	Java Data Objects
JEE	–	Java Enterprise Edition
JPA	–	Java Persistance API
JPOX	–	Java Persistent Objects
LGPL	–	Lesser General Public License
NIST	–	National Institute of Standards and Technology
ORM	–	Object Relational Mapping, Objektově-relační mapování
SQL	–	Structured Query Language, Strukturovaný dotazovací jazyk
SŘBD	–	Systém řízení báze dat
URL	–	Uniform Resource Locator , jednotný lokátor zdrojů
W3C	–	World Wide Web Consortium
XML	–	Extensible Markup Language

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Informační systém je systém pro získávání, zpracovávání, uchovávání a předávání informací. Informace tvoří kódovaná data, která lze uchovávat a zpracovávat technickými prostředky. Tyto data je nutné někde bezpečným způsobem ukládat. K tomu slouží datová vrstva informačního systému.

V současné době většina organizací a firem potřebuje shromažďovat a uchovávat velké množství informací. Ať už se jedná o seznamy zákazníků, faktury, inventární majetek anebo výplaty zaměstnanců. Proto existuje velké množství softwarových firem, které se vývojem informačních systému zabývají. Tvorba těchto systému však není jednoduchá záležitost. Je to poměrně komplikovaný proces, procházející od analýzy a návrhu až po implementaci a uživatelskou podporu vytvořeného systému.

Pro tvorbu informačních systému se většinou používá některého z objektově orientovaných programovacích jazyků v kombinaci s některým relačním SŘBD. Proto je nutné vyřešit, jakým způsobem se bude vytvořený objekt ukládat do relační databáze. Technologie řešící tento problém, se nazývá objektově-relační mapování. Tato diplomová práce se zabývá objektově-relačním mapováním na platformě JAVA.

Kapitola 2 popisuje pojem perzistence dat a probírá možnosti jejího využití při tvorbě informačních systémů. Seznamuje čtenáře s různými úrovněmi perzistence dat a upřesňuje oblasti jejich použití. Popisuje jakými způsoby mohou být data ukládána. Prezentuje také možnosti serializace v objektově orientovaném programovacím jazyce JAVA. V poslední části se věnuje typům databází a popisuje jejich historii.

Kapitola 3 je zaměřena na objektově-relační mapování. Seznamuje čtenáře s důvody vzniku ORM a prezentuje vybrané standardy v této oblasti. Popisuje rovněž způsoby určování kvality objektově-relačního mapování . Také jsou zde uvedeny doporučené systémové požadavky pro realizaci ORM.

Kapitola 4 popisuje vztahy mezi objektovým a relačním modelem. Ukazuje, jakým způsobem jsou objekty ukládány do relační databáze, popisuje možné vztahy mezi těmito objekty a způsoby realizace těchto vztahů v relační databázi. V závěru seznamuje čtenáře s možnostmi definování mapování objektů do tabulek v relační databázi.

Kapitola 5 se věnuje ORM nástrojů využívaných při tvorbě informačních systémů. Popisuje důvody jejich vzniku a podrobněji prezentuje dva vybrané frameworky. Jedná se o ORM frameworky Hibernate a EclipseLink. Obsahuje zhodnocení použití ORM frameworků a upozorňuje na možná úskalí při jejich implementaci.

Kapitola 6 popisuje analýzu a návrh ORM frameworku EasyORM. Jsou v ní uvedeny jednotlivé požadavky na funkčnost frameworku a podrobná analýza těchto požadavků. Je v ní proveden návrh vytvářeného frameworku a jsou zobrazeny jeho modely.

Kapitola 7 popisuje implementaci ORM frameworku EasyORM. Probírá se v ní výběr technologických prostředků a popisuje způsob implementace.

Kapitola 8 definuje metody porovnávání vybraných ORM frameworků. Je v ní popsán způsob měření výkonu jednotlivých ORM frameworků a jsou v ní prezentovány výsledky porovnávání.

V závěru jsou zhodnoceny přínosy této práce.

2 Perzistence dat

Tato kapitola popisuje pojem perzistence dat a probírá možnosti jejího využití při tvorbě informačních systémů. Seznamuje čtenáře s různými úrovněmi perzistence dat a upřesňuje oblasti jejich použití. Popisuje jakými způsoby mohou být data ukládána. Prezentuje také možnosti serializace v objektově orientovaném programovacím jazyce JAVA. V poslední části se věnuje typům databází a popisuje jejich historii.

2.1 Teorie

Každý informační systém vytváří nebo shromažďuje informace. Pokud by tyto informace existovali pouze v operační paměti, tak by se po vypnutí serveru tyto data (informace) ztratili. Musí být tedy nějakým způsobem zajištěno uložení těchto dat tak, aby je bylo možné později znovu načíst do operační paměti. Tento mechanizmus se nazývá perzistence dat. Data se tedy nazývají perzistentní, pokud přetrvávají i po ukončení aplikace, která je vytvořila. Perzistentní data se rozdělují podle NIST [1] do tří úrovní:

- částečně perzistentní data – je uložena pouze aktuální hodnota dat
- perzistentní data – je uložena aktuální hodnota dat a hodnota před poslední změnou
- plně perzistentní data – jsou uloženy všechny hodnoty dat od jejich vzniku

První úroveň je dostatečná pro tvorbu běžných aplikací. Rozumně naprogramovaný informační systém, ale pracuje minimálně s úrovní „perzistentní data“. To mu umožňuje v případě výpadku systému obnovit konzistentní stav dat pomocí volání „rollback“. Tuto úroveň podporují také všechny běžně používané SŘBD. Proto je výhodné, když při tvorbě informačních systému, použijeme pro ukládání dat nějaký SŘBD. Tím se ušetří spousta práce se zajištěním perzistence dat. Třetí úroveň se používá minimálně, například v systémech pro správu verzí (CVS, Subversion).

2.2 Ukládání dat

Již od počátku programování se řeší problém kam bezpečně ukládat data. Data se mohou uložit několika způsoby. Nejčastěji se používá uložení dat do souboru nebo databáze. Pokud se budou data ukládat do souboru, musí se vyřešit problém s formátem ukládání dat. V objektově orientovaném programování, jsou data většinou obsažená v atributech objektů. Tyto objekty pak mají různou strukturu v závislosti na tom, jakou entitu z reálného světa představují. V objektově orientovaném programovacím jazyce Java, je možné využít serializaci k uložení objektů do souboru. Jestliže se budou data ukládat do databáze, je nutné vybrat, jaký typ SŘBD bude použit. Pokud se nebude používat objektový SŘBD, tak se musí ještě vyřešit, jakým způsobem se budou objekty do databáze ukládat. Jak ukládání dat do souboru, tak do databáze, má své výhody a nevýhody.

Výhody ukládání dat do souboru:

- datové struktury jsou navrženy přímo pro danou aplikaci
- nízká hardwarová náročnost

Nevýhody ukládání dat do souboru:

- závislost datových struktur na aplikaci
- komplikovaná implementace zamykání záznamů při vícenásobném přístupu
- chybějící podpora transakcí
- komplikované řešení přístupových práv
- chybějící integritní omezení

Výhody použití databází:

- přímý přístup k požadovaným datům
- vyšší rychlost přístupu
- možnost paralelního přístupu
- propracovaný systém přístupových práv
- možnost vyhledání dat pomocí SQL dotazů
- podpora transakcí

Nevýhody použití databází:

- komplikovaná implementace složitých datových struktur
- komplikovaná implementace nestandardního přístupu k datům

2.3 Serializace

Pojmem serializace se označuje vytvoření proudu symbolů, které představují vnitřní stav objektu [2, 3]. Proud symbolů musí být vytvořen tak, aby bylo možné po načtení tohoto proudu, znovu obnovit vnitřní stav objektu. Tomuto procesu se říká deserializace. Proud symbolů může být uložen do souboru, nebo prostřednictví síťového protokolu odeslán na jiný počítač. Tato podkapitola se zabývá binární a XML serializací v objektovém orientovaném programovacím jazyce JAVA. Popisuje jejich výhody a nevýhody a prezentuje možnosti využití.

2.3.1 Binární serializace

Při binární serializaci je vnitřní stav objektu uložen jako proud bajtů. Předpokladem pro provádění serializace v Javě je implementace rozhraní java.io.Serializable. Při implementaci tohoto rozhraní není nutné překrývat nějaké metody. Pouze ve výjimečných případech je nutné překrýt metody writeObject() a readObject().

Výhody:

- zjednodušení programového kódu
- jednoduché ukládání objektů

Nevýhody:

- serializované data nelze jednoduše načítat jiným programovacím jazykem
- při změně struktury objektu není možné načíst uložené serializované data
- při načítaní složených objektů, je nutné načíst celý objekt

2.3.2 XML serializace

Odstraňuje hlavní nevýhody binární serializace. Data jsou ukládána do formátu XML, čímž je umožněna přenositelnost mezi různými programovacími jazyky. XML je obecný značkovací jazyk vyvinutý konsorciem W3C [4]. Byl vyvinut především pro výměnu dat mezi aplikacemi a pro publikování dokumentů.

2.4 Databáze

Zjednodušeně si lze databázi představit jako úložiště dat, tedy určitý prostor obsahující data. Je to tedy místo do kterého je možné ukládat data. Přístup k datům zajišťuje software, který se nazývá „Systém řízení báze dat“, zkráceně SŘBD. Tento název vzniknul z anglického „database management systém“ [5, 6, 7].

Historie databází sahá až do 19. století. Již v roce 1890 vytvořil Herman Hollerith první automat na bázi děrných štítků pro státní úřady USA. Později v roce 1911 došlo k sloučení jeho firmy s další a vznikla firma IBM. V roce 1935 vytvořila firma IBM první digitální počítač pro komerční využití UNIVAC I. V roce 1960 bylo ministerstvem obrany USA vytvořeno seskupení Data Systems Languages, jenž vytvořilo programovací jazyk COBOL.

V roce 1961 Charles Bachman z General Electric představil první integrovaný datový sklad s databázovým managementem. V roce 1970 Ted Codd publikoval článek „A Relational Model of Data for Large Shared Data Banks“, což byl návrh na implementaci nového datového modelu, který byl nazván „relačním“. Představil v něm možnosti využití relačního kalkulu a algebry pro ukládání a manipulaci s daty. Data měla být ukládána do tabulek.

V roce 1976 byl dokončen dotazovací jazyk SEQUEL2, který byl později přejmenován na SQL. V roce 1980 firma Oracle představila první SQL databázi. V roce 1985 vznikl projekt Postgres, jehož cílem bylo vytvořit relačně-objektovou databázi. Později byl přejmenován na Postgres95 a po přechodu na open source licenci byl znovu přejmenován na PostgreSQL. Pod tímto názvem je vyvíjen dodnes.

2.4.1 Relační databáze

První relační databáze vznikaly již v 80. letech minulého století. Od té doby prošly dlouhým vývojem a jsou velmi dobře uzpůsobeny pro ukládání a manipulaci velkého množ-

ství dat. Tyto databáze jsou navíc poměrně jednoduché, snadno pochopitelné a pro většinu aplikací plně dostačující. V současné době jsou proto nejrozšířenějším typem databází.

Základem tohoto typu databází je relační algebra, což je pojem označující jazyk vysoké úrovně, zabývající se manipulací s relacemi. Základními operacemi tohoto jazyka jsou projekce, selekce a spojení. Tyto databáze tedy používají relační model dat [7]. Data jsou uspořádány do tabulek. Sloupce tabulek se nazývají atributy a řádky označujeme jako záznamy. Množinu všech hodnot, které může atribut obsahovat, nazýváme doména atributu. Tabulka pak realizuje podmnožinu kartézského součinu možných dat všech sloupců - relaci.

Výhody:

- standardní dotazovací jazyk SQL
- vyhledání a modifikace dat pomocí SQL
- možnost definování integritních omezení
- podpora transakcí
- rychlost přístupu k datům

Nevýhody:

- nelze získat samostatné data, výsledkem je vždy záznam
- komplikovaná implementace složitých datových struktur, které se musí rozložit na více tabulek
- komplikovaná implementace dat s proměnlivou délkou

2.4.2 Objektové databáze

Objektově orientované databáze začali vznikat ve druhé polovině 80. let, na základě potřeby uchovávat a pracovat s objekty vytvořenými v objektově orientovaných programovacích jazycích. Při použití těchto databází není nutné převádět data z objektové podoby do relační, data jsou ukládána přímo ve formě objektů. Tyto databáze umožňují využít všech výhod objektově orientovaného programování. Umožňují použití dědičnosti, polymorfizmu a referencí na jiné objekty [8, 9]. V případě složitých datových struktur, je přístup k datům rychlejší než u relačních databází. Objekt je vyhledán přímo pomocí reference, není nutné data potřebná k vytvoření objektu vyhledávat ve více tabulkách. Proto pro aplikace, které vyhledávají objekty převážně na základě referencí, je použití těchto databází výhodné. V případě vyhledávání objektů podle jejich atributů, je vyhledávání podstatně pomalejší než u relačních databází.

Výhody:

- ukládají se objekty
- lepší podpora datových struktur
- obsahuje vazby mezi objekty

- snadná rozšiřitelnost pomocí dědičnosti
- efektivnější zpracování dotazů

Nevýhody:

- nedostupnost těchto databází a jejich vysoká cena
- nedostatečné dodržování standardů ODMG
- nedostatek praktických zkušeností s těmito databázemi
- nedostatečná podpora metod analýzy a návrhu

2.4.3 Objektově-relační databáze

Tyto databáze používají objektově relační datový model. Jedná se o klasický relační datový model, který je rozšířen o podporu datových struktur, který je znám z oblasti objektově orientovaných programovacích jazyků [10]. Jedná se o struktury vzniklé kombinací základních datových typů. Většina velkých relačních databází podporuje tento datový model. Objektově relační databáze, ale stále zůstává ve svých základech relační databázi.

2.4.4 Postrelační databáze

Postrelační databáze většinou mají integrovaný modul řešící objektově-relační mapování. To jim umožňuje dosáhnout vysokého výkonu jak u relačního, tak i u objektového přístupu k datům. Typickým příkladem těchto databází je databáze Caché od společnosti InterSystem Corporation [11, 12]. Caché ukládá data ve vícerozměrných strukturách, které lépe popisují data ze skutečného světa. Přístup k těmto datům je potom mnohem rychlejší. Databáze Caché také nabízí několik variant přístupu k datům. Můžeme přistupovat k datům prostřednictvím jazyka SQL, stejně jako v objektové databázi, vícerozměrným přístupem k datovým polím anebo pomocí HTML. Všechny tyto varianty přístupu lze použít zároveň, i při současném přístupu k jedné položce. Tyto dva faktory, vícerozměrné datové struktury a více variant přístupu k datům, jsou hlavními znaky postrelačních databází. Jedná se v podstatě o spojení objektových a relačních databází.

3 Objektově-relační mapování

Tato kapitola je zaměřena na objektově-relační mapování. Seznamuje čtenáře s důvody vzniku ORM a prezentuje vybrané standardy v této oblasti. Popisuje rovněž způsoby určování kvality objektově-relačního mapování. Také jsou zde uvedeny doporučené systémové požadavky pro realizaci ORM.

3.1 Teorie

Objektově relační mapování je programovací technika provádějící konverzi dat mezi objektovým a relačním datovým modelem [13]. V objektově orientovaném programování se většinou pracuje s objekty představujícími nějaké entity z reálného nebo virtuálního světa. V případě že bude nutné ukládat tyto objekty do relační databáze, tak se musí převést do dat v relační formě (a naopak). Vzhledem k tomu, že objektově orientovaný návrh dat nelze přímo převést na relační model (a naopak), je nutné použít ORM. Při načítaní dat z relační databáze musí ORM zajistit nastavení příslušných atributů objektů. Naopak v případně ukládání objektů, musí ORM převést atributy objektů na data v relační formě. Snahou ORM je využití výhod obou těchto technologií tak, aby objekty představovaly entity reálného světa a zároveň bylo možné využívat všech výhod relačních databází.

Výhody ORM:

- pracuje se s objektovým modelem
- rychlejší vytváření aplikací
- přenositelnost mezi různými SŘBD
- typová kontrola
- nevznikají chyby v SQL
- jednodušší testování

Nevýhody ORM:

- menší výkon aplikace
- nevyužívá všechny možnosti SŘBD
- přístup do databáze je realizován pod jedním společným účtem

3.2 Systémové požadavky

Používat objektově-relační mapování je možné ve všech objektově orientovaných programovacích jazycích. Také data mohou být ukládána do libovolné relační databáze. Pro smysluplné použití ORM je ale vhodné, aby byly splněny alespoň některé systémové požadavky.

Systémové požadavky:

- objektově orientovaný programovací jazyk
- práce s meta-informacemi
- přístup do databáze pomocí ODBC nebo JDBC
- databáze musí podporovat standard ANSI SQL

Jako velmi vhodné se jeví použití platformy JAVA nebo DOT.NET v kombinaci s relační databází podporující standard ANSI SQL.

3.3 Standardy objektově-relačního mapování

V oblasti objektově-relačního mapování se používá několik standardů. Zde jsou prezentovány nejdůležitější dva s těchto standardů.

3.3.1 Java Data Objects

Java Data Objects je standard objektově orientovaného programovacího jazyka Java, který se zabývá perzistencí objektů [14]. Java Data Objects je pouhá specifikace rozhraní mezi objekty jazyka Java a úložišti perzistentních dat. Pro vývoj projektu se pak musí použít nějaká konkrétní implementace JDO. Perzistentní úložiště nemusí být pouze SŘBD, ale může to být také XML soubor anebo dokonce obyčejný textový soubor.

Na tvorbě tohoto standardu se podílelo mnoho významných společností zabývajících se informačními technologiemi. První verze JDO 1.0 byla vyvíjená pod označením „JSR 12“. Verze JDO 2.0 byla vyvíjená pod označením „JSR 243“ a byla dokončena v roce 2006. V současné době je k dispozici verze JDO 2.2 a v brzké době by měla být uvolněna verze JDO 2.3. Záměrem tohoto standardu je oddělit aplikační logiku od konkrétního úložiště perzistentních dat. Proto je možné změnit používanou implementaci JDO, bez nutnosti velkých úprav ve zdrojovém kódu projektu.

Existuje velké množství různých implementací JDO. Úroveň těchto implementací je velmi rozdílná a jednotlivé implementace většinou nejsou stoprocentně kompatibilní. V současné době je nejsilnější podpora relačních databází. Některé implementace se orientují pouze na relační databáze a umožňují mapování Java objektů na vybranou množinu SŘBD.

3.3.2 Java Persistence API

Java Persistence API (JPA) je standard objektově orientovaného programovacího jazyka Java, který umožňuje objektově relační mapování (ORM) [15, 16]. Úkolem tohoto standardu je usnadnění práce s ukládáním objektů do databáze a naopak. Je součástí vývojového prostředí Java Enterprise Edition, ale je možné jej použít i ve vývojovém prostředí Java Standard Edition. Java Persistence API je pouhá specifikace rozhraní pro práci s ORM frameworky. Pro vývoj projektu se pak musí použít nějaký ORM framework, který má implementován toto rozhraní. Při vlastním programování projektů se používá rozhraní Java Persistence API, což umožňuje změnit používáný ORM framework, bez nutnosti úprav ve zdrojovém kódu projektu.

3.4 Kvalita objektově-relačního mapování

Objektově-relační mapování může být implementováno různými způsoby. Jeden z předních světových odborníků v oblasti objektově-relačního mapování Mark Fussel, definoval následující čtyři úrovně kvality [18].

3.4.1 Čistě relační

Celá aplikace včetně uživatelského rozhraní je založena na relačním modelu a využívání operací SQL. Tento přístup není vhodný pro velké systémy, ale pro jednoduché aplikace může být plně dostačující. Kód SQL může být přesně optimalizován pro potřeby těchto aplikací. Nevýhodou je nízká přenositelnost a udržovatelnost těchto aplikací. Tyto aplikace často využívají uložené procedury a přenášejí tak část funkcionality z byznys vrstvy do databáze.

3.4.2 Lehce objektové

Entity jsou reprezentovány třídami, které jsou ručně mapovány na relační tabulky. SQL kód je psaný ručně a využívá JDBC. Pomocí návrhových vzorů je oddělen od byznys logiky. Tento přístup je poměrně hodně rozšířený a je úspěšně používán především u aplikací s malým počtem entit. Také v této úrovni je možné použít uložené procedury.

3.4.3 Středně objektové

K návrhu aplikace je použitý objektový model. SQL kód je vytvářen při překladu aplikace, nebo je generován přímo v aplikaci. Vztahy mezi objekty jsou podporovány perzistentním mechanismem aplikace. Dotazy jsou vytvářeny pomocí objektově-orientovaného jazyka. Objekty jsou uchovávány ve vyrovnávací paměti perzistentní vrstvy. Velké množství ORM produktů patří minimálně do této úrovně. Tento přístup je vhodný pro středně velké aplikace využívající transakce, především pokud chceme využívat více SŘBD. Tyto aplikace již většinou nepoužívají uložené procedury.

3.4.4 Plně objektové

Tato úroveň podporuje plné objektové mapování zahrnující agregaci, dědičnost, polymorfizmus a perzistenci dle dosažitelnosti. Perzistentní vrstva implementuje transparentní perzistenci, proto perzistentní třídy nemusí mít společného předka, ani nemusí implementovat žádné rozhraní. Aplikace má navíc implementovány efektivní strategie pro využití vyrovnávací paměti. Řada ORM produktů již dosáhla této úrovně kvality.

4 Mapování objektů do databáze

Tato kapitola popisuje vztahy mezi objektovým a relačním modelem. Ukazuje, jakým způsobem jsou objekty ukládány do relační databáze, popisuje možné vztahy mezi těmito objekty a způsoby realizace těchto vztahů v relační databázi. V závěru seznamuje čtenáře s možnostmi definice mapování objektů do tabulek v relační databázi [23, 18].

4.1 Entita

Pod pojmem entita se v objektově-relačním mapování rozumí určitý objekt, který je potřeba perzistentně uložit do relační databáze. Toto uložení může být realizováno několika způsoby. Nejvhodnější způsob se určí podle typu entity. V případě jednoduché entity je možné jí uložit, jako jeden záznam v jedné tabulce. U složitější entity může její uložení představovat i několik záznamů ve více databázových tabulkách. Entitu v objektově orientovaných programovacích jazycích, většinou reprezentuje instance konkrétní třídy.

Názorným příkladem jednoduché entity je entita $Uzivatel$, která představuje uživatele IS. Výpis kódu 1 ukazuje třídu, jejíž instance reprezentuje tuto entitu v objektově orientovaném programovacím jazyce Java.

```
public class Uzivatel
{
    private int id;
    private String jmeno;
    private String prijmeni;
    private String email;

    public String getEmail()
    {
        return email;
    }

    public void setEmail(String email)
    {
        this.email = email;
    }

    ......
    ......
}
```

Výpis 1: Java – implementace jednoduché entity

Ve výpisu 2 je zobrazen SQL skript, který vytvoří tabulku $Uzivatel$ v relační databázi. Jediná tabulka bude použita pro uložení entity.

```
CREATE TABLE Uzivatel
(
    id          INT NOT NULL PRIMARY KEY auto_increment,
    jmeno       VARCHAR(20) NOT NULL,
    prijmeni    VARCHAR(20) NOT NULL,
    email       VARCHAR(50) NOT NULL
```

```
);
```

Výpis 2: SQL skript – jednoduchá entita – 1 tabulka

Entita ale také může být uložena do dvou databázových tabulek. Ve výpise 3 je zobrazen SQL skript, který tyto tabulky vytvoří.

```
CREATE TABLE Uzivatel
(
   id              INT NOT NULL PRIMARY KEY,,
   jmeno           VARCHAR(20) NOT NULL,
   prijmeni        VARCHAR(20) NOT NULL
);

CREATE TABLE Uzivatel_kontakt
(
   id              INT NOT NULL PRIMARY KEY,
   email           VARCHAR(50) NOT NULL
);
```

Výpis 3: SQL skript – jednoduchá entita – 2 tabulky

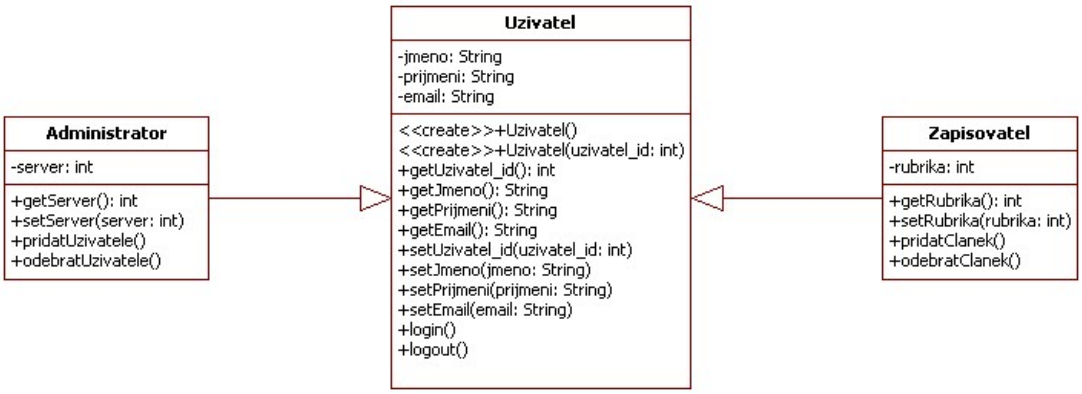
Původní entita je pak získána spojením záznamů z obou tabulek pomocí stejného primárního klíče id.

4.2 Dědičnost

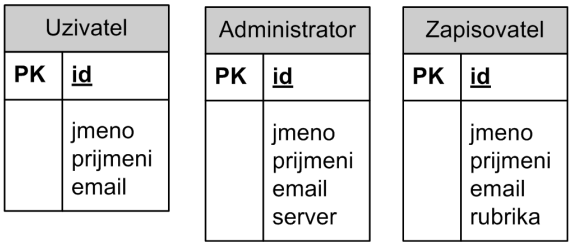
Dědičnost je jedním z nejdůležitějších principů objektově orientovaného programování. Pomocí dědičnosti mohou třídy získávat atributy a metody od jiných tříd. Říká se, že potomek zdědí vlastnosti rodičovské třídy. Tyto atributy a metody si může v případě potřeby také vhodně upravit. Díky tomu není nutné programovat stejný kód ve více třídách. Proto je vytvářený zdrojový kód přehlednější a lépe modifikovatelný. Java podporuje pouze jednoduchou dědičnost. Znamená to, že třída může získat vlastnosti pouze jedné rodičovské třídy. Relační databáze bohužel nemají implementovanou podporu dědičnosti. Proto je nutné tuto vlastnost implementovat pomocí databázových tabulek a vazeb mezi nimi.

Názorným případem dědičnosti je tento jednoduchý demonstrační model. V informačním systému se vyskytuje několik typů uživatelských rolí. Běžný uživatel může pouze číst publikované informace. Zapisovatel může vkládat informace do určené rubriky. Administrátor spravuje uživatelské účty na přiřazeném serveru. Tyto role mají většinu vlastností stejných. Proto je vhodné použít výhod dědičnosti. Třída $Uzivatel$ obsahuje všechny společné vlastnosti. Třídy $Administrator$ a $Zapisovatel$ pak obsahují pouze specifické vlastnosti jednotlivých rolí. Na obrázku 1 je tato situace znázorněna pomocí diagramu tříd.

Existuje několik způsobů, kterými je možné implementovat dědičnost do relační databáze.



Obrázek 1: Diagram tříd – dědičnost



Obrázek 2: ER diagram – horizontální mapování dědičnosti

4.2.1 Horizontální mapování

Při tomto mapování existuje samostatná tabulka pro každou jednotlivou třídu. V každé tabulce jsou uloženy všechny atributy dané třídy a to i zděděné atributy. Na obrázku 2 je konceptuální model znázorňující rozložení atributů třídy do databázových tabulek.

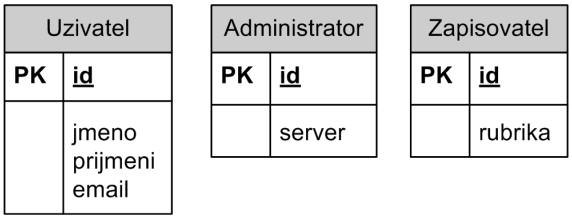
Každá tabulka má samostatný primární klíč id. Tento primární klíč slouží k jednoznačné identifikaci každého záznamu v tabulce. Tabulka $Uzivatel$ obsahuje všechny atributy nadtřídy $Uzivatel$. Tabulka $Administrator$ obsahuje atribut $server$ z podtřídy $Administrator$ a také všechny atributy nadtřídy $Uzivatel$. Tabulka $Zapisovatel$ obsahuje atribut $rubrika$ z podtřídy $Zapisovatel$ a také všechny atributy nadtřídy $Uzivatel$.

Výhody:

- jednoduché a rychlé dotazy na entity konkrétního typu

Nevýhody:

- složité dotazy nad obecnými entitami (nadtřída a podtřídy)
- problematické vazby na obecné entity
- neexistuje jedinečný primární klíč nad obecnými entitami



Obrázek 3: ER diagram – vertikální mapování dědičnosti

- při změně atributů nadtřídy musíme upravit všechny tabulky
- velký počet tabulek

4.2.2 Vertikální mapování

Také při tomto mapování existuje samostatná tabulka pro každou jednotlivou třídu. V každé tabulce jsou však uloženy pouze atributy dané třídy. Toto mapování je velmi podobné diagramu tříd a implementaci ve zdrojovém kódu Java. Na obrázku 3 je konceptuální model znázorňující rozložení atributů tříd do databázových tabulek.

Tabulka $Uzivatel$ má jedinečný primární klíč id. U tabulek $Administrator$ a $Zapisovatel$ je primární klíč zároveň i cizím klíčem odkazujícím na tabulku $Uzivatel$. V případě ukládání entity třídy $Uzivatel$ se vytvoří pouze jediný záznam v tabulce $Uzivatel$. Pokud, ale ukládáme entitu některé podtřídy, tak se vytvoří záznam, jak v tabulce $Uzivatel$, tak i v tabulce dané podtřídy. Oba záznamy budou mít stejný primární klíč id.

Výhody:

- flexibilita
- při změně atributů nadtřídy upravujeme pouze jednu tabulku

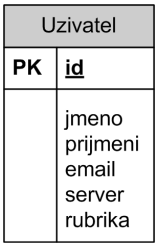
Nevýhody:

- složité dotazy nad entitami podtříd
- pomalejší načítání entit podtříd
- velký počet tabulek
- při změně atributů nadtřídy musíme upravit všechny tabulky

4.2.3 Mapování pomocí unie

Při tomto mapování existuje jediná tabulka pro všechny třídy. V této tabulce jsou uloženy jak všechny atributy dané nadtřídy, tak i všechny atributy podtříd. Na obrázku 4 je konceptuální model znázorňující rozložení atributů tříd do databázové tabulky.

Tabulka $Uzivatel$ má jedinečný primární klíč id. Tabulky $Administrator$ a $Zapisovatel$ se při tomto mapování nepoužívají. Vzniká ale problém, jak jednoduše zjistit, který typ entity jednotlivé záznamy reprezentují. Proto se většinou přidává do tabulky atribut, který jednoznačně určuje typ entity.



Obrázek 4: ER diagram – mapování dědičnosti pomocí unie

Výhody:

- jediná tabulka

Nevýhody:

- atribut v tabulce určující typ entity
- při větším počtu podtříd vysoké prostorové nároky
- problém s integritními omezeními u atributů podtříd

4.3 Vztahy mezi entitami

Jednou z nejdůležitějších věcí, kterou je nutné v objektově relačním mapování implementovat, jsou vztahy mezi jednotlivými entitami. V objektově orientovaných programovacích jazycích se vztahy implementují tak, že třída obsahuje atribut s referencí na jinou třídu. Výpis kódu 4 představuje ukázkový příklad, implementace vztahů mezi instancemi tříd v Javě.

```
public class Uzivatel
{
    private int  uzivatel_id ;
    private String  jmeno;
    private String  prijmeni ;
    private String  email;
    private Ucet ucet;

     .....
     .....
}

public class Ucet
{
    private int  ucet_id ;
    private int  cislo ;
    private int  kod;

     .....
     .....
```

```
}
```

Výpis 4: Java – vztahy mezi entitami

Ve výpisu 5 je zobrazena implementace těchto vztahů v relační databázi.

```
CREATE TABLE Ucet
(
    ucet_id          INT NOT NULL PRIMARY KEY auto_increment,
    cislo            INT NOT NULL,
    kod              INT NOT NULL
);

CREATE TABLE Uzivatel
(
    uzivatel_id      INT NOT NULL PRIMARY KEY auto_increment,
    jmeno            VARCHAR(20) NOT NULL,
    prijmeni         VARCHAR(20) NOT NULL,
    email            VARCHAR(50) NOT NULL,
    ucet             INT NOT NULL,
    FOREIGN KEY (ucet) REFERENCES Ucet (ucet_id)
);
```

Výpis 5: SQL skript – vztahy mezi entitami

Odkaz na tabulku $Ucet$ se vytvoří tak, že se uloží její primární klíč $ucet_id$ jako cizí klíč $ucet$ v tabulce $Uzivatel$. Tabulka $Ucet$ musí být definována před tabulkou $Uzivatel$, jinak dojde k chybě při zpracování SQL skriptu SŘBD. První se vytváří tabulky, na které budeme odkazovat. U některých SŘBD je sice opačné pořadí možné, ale tento postup nelze obecně doporučit.

4.3.1 Vztah typu 1:1

Tento typ vztahu popisuje velice primitivní vztah mezi dvěma entitami. Entita může být odkazována pouze jedinou entitou. Ukázkový příklad se může upravit na tento typ násobnosti vztahu dvěma způsoby.

4.3.1.1 Mapování pomocí primárního klíče Entity s touto vazbou mají stejnou hodnotu primárního klíče. Primární klíč druhé tabulky je také cizím klíčem odkazujícím na první tabulku. Ve výpisu 6 je zobrazena implementace tohoto vztahu v relační databázi.

```
CREATE TABLE Ucet
(
id                   INT NOT NULL PRIMARY KEY,
cislo                INT NOT NULL,
kod                  INT NOT NULL
);

CREATE TABLE Uzivatel
(
id                   INT NOT NULL PRIMARY KEY,
jmeno                VARCHAR(20) NOT NULL,
```

```
prijmeni            VARCHAR(20) NOT NULL,
email               VARCHAR(50) NOT NULL,
ucet                INT NOT NULL,
FOREIGN KEY (id) REFERENCES Ucet (id)
);
```

Výpis 6: SQL skript – vztah typu 1:1 pomocí primírního klíče

4.3.1.2 Mapování pomocí cizího klíče

Entity s touto vazbou nemají stejnou hodnotu primárního klíče. Druhá tabulka obsahuje cizí klíč, odkazující na první tabulku. Aby se jednalo o vztah typu 1:1, je nutné, aby tento cizí klíč byl unikátní. Ve výpisu 7 je zobrazena implementace tohoto vztahu v relační databázi.

```
CREATE TABLE Ucet
(
   ucet_id          INT NOT NULL PRIMARY KEY auto_increment,
   cislo            INT NOT NULL,
   kod              INT NOT NULL
);

CREATE TABLE Uzivatel
(
    uzivatel_id     INT NOT NULL PRIMARY KEY auto_increment,
   jmeno            VARCHAR(20) NOT NULL,
   prijmeni         VARCHAR(20) NOT NULL,
   email            VARCHAR(50) NOT NULL,
   ucet             INT UNIQUE NOT NULL,
   FOREIGN KEY (ucet) REFERENCES Ucet (ucet_id)
);
```

Výpis 7: SQL skript – vztah typu 1:1 pomocí cizího klíče

4.3.2 Vztah typu 1:N

Tento typ vztahu popisuje jednoduchý vztah mezi dvěma entitami. Každá entita může odkazovat na jednu nebo více entit. Také, ale nemusí odkazovat na žádnou entitu. Ve výpisu 8 je zobrazena implementace tohoto vztahu v Javě.

```
public class Uzivatel
{
    private int  uzivatel_id ;
    private String  jmeno;
    private String  prijmeni ;
    private String  email;
    private List<Ucet> ucet = new ArrayList<Ucet>();

     .....
     .....
}

public class Ucet
```

```
{
    private int ucet_id;
    private int cislo;
    private int kod;

     .....
     .....
}
```

Výpis 8: Java – implementace vztahu typu 1:N

Ve výpisu 9 je zobrazena jedna z možných implementací tohoto vztahu v relační databázi.

```
CREATE TABLE Ucet
(
   ucet_id          INT NOT NULL PRIMARY KEY auto_increment,
   cislo            INT NOT NULL,
   kod              INT NOT NULL
);

CREATE TABLE Uzivatel
(
    uzivatel_id     INT NOT NULL PRIMARY KEY auto_increment,
   jmeno            VARCHAR(20) NOT NULL,
   prijmeni         VARCHAR(20) NOT NULL,
   email            VARCHAR(50) NOT NULL,
   ucet             INT NULL,
   FOREIGN KEY (ucet) REFERENCES Ucet (ucet_id)
);
```

Výpis 9: SQL skript – vztah typu 1:N

4.3.3 Vztah typu N:1

Tento typ vztahu popisuje jednoduchý vztah mezi dvěma entitami. Několik entit může odkazovat na jednu entitu. Také, ale na ní nemusí odkazovat žádná entita. Ve výpisu 10 je zobrazena implementace tohoto vztahu v Javě.

```
public class Uzivatel
{
    private int  uzivatel_id;
    private String  jmeno;
    private String  prijmeni;
    private String  email;
    private Ucet ucet;

     .....
     .....
}

public class Ucet
{
```

```
    private int ucet_id;
    private int cislo;
    private int kod;

     .....
     .....
}
```

Výpis 10: Java – implementace vztahu typu N:1

Ve výpisu 11 je zobrazena jedna z možných implementací tohoto vztahu v relační databázi.

```
CREATE TABLE Ucet
(
   ucet_id          INT NOT NULL PRIMARY KEY auto_increment,
   cislo            INT NOT NULL,
   kod              INT NOT NULL
);

CREATE TABLE Uzivatel
(
   uzivatel_id      INT NOT NULL PRIMARY KEY auto_increment,
   jmeno            VARCHAR(20) NOT NULL,
   prijmeni         VARCHAR(20) NOT NULL,
   email            VARCHAR(50) NOT NULL,
   ucet             INT NOT NULL,
   FOREIGN KEY (ucet) REFERENCES Ucet (ucet_id)
);
```

Výpis 11: SQL skript – vztah typu N:1

4.3.4 Vztah typu M:N

Tento typ vztahu popisuje složitější vztah mezi dvěma entitami. Každá entita může odkazovat na jednu nebo více entit. Také ale nemusí odkazovat na žádnou entitu. A zároveň několik entit, může odkazovat na jednu entitu. Také, ale na ní nemusí odkazovat žádná entita. Typickým příkladem může být členství uživatelů v diskusních fórech. Jeden uživatel může být členem několika diskusních fór. Také, ale nemusí být členem žádného diskusního fóra. Stejně tak diskusní fórum může mít několik členů, ale také nemusí mít žádného uživatele. V následujícím výpisu 12 je zobrazena implementace tohoto vztahu v Javě.

```
public class Uzivatel
{
    private int uzivatel_id;
    private String jmeno;
    private String prijmeni;
    private String email;
    private List<Forum> fora = new ArrayList<Forum>();

     .....
```

```
    .....
}

public class Forum
{
    private int forum_id;
    private String nazev;
    private List<Uzivatel> uzivatele = new ArrayList<Uzivatel>();

    .....
    .....
}
```

Výpis 12: Java – implementace vztahu typu M:N

Ve výpisu 13 je zobrazena implementace tohoto vztahu v relační databázi. Je při ní využito vazební tabulky $UzivateleVeForu$, ve které jsou uloženy vztahy mezi entitami.

```
CREATE TABLE Forum
(
   forum_id          INT NOT NULL PRIMARY KEY auto_increment,
   nazev             VARCHAR(50) NOT NULL
);

CREATE TABLE Uzivatel
(
   uzivatel_id       INT NOT NULL PRIMARY KEY auto_increment,
   jmeno             VARCHAR(20) NOT NULL,
   prijmeni          VARCHAR(20) NOT NULL,
   email             VARCHAR(50) NOT NULL,
);

CREATE TABLE UzivateleVeForu
(
   uzivatel          INT NOT NULL,
   forum             INT NOT NULL,
   FOREIGN KEY (uzivatel) REFERENCES Uzivatel (uzivatel_id)
   FOREIGN KEY (forum) REFERENCES Forum (forum_id)
);
```

Výpis 13: SQL skript – vztah typu M:N

4.4 Definice mapování

Seznámili jsme se s možnostmi mapování objektů na tabulky v relační databázi. Toto mapování, se ale musí nějakým způsobem definovat. Většina ORM frameworků nabízí několik možností pro definování tohoto mapování. Zde jsou uvedené způsoby které používá ORM framework Hibernate [17, 18].

4.4.1 Externí mapovací soubory

Definice ORM je uložena v externím souboru. Obvykle se používá XML soubor. Pomocí tohoto souboru jsou vygenerovány perzistentní třídy a je vytvořeno odpovídající databázové schéma. Externí soubor popisuje, do kterého sloupce, a jaké databázové tabulky budou jednotlivé atributy třídy uloženy. Na výpisu 14 je ukázka mapovacího XML souboru.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://
    hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 14.4.2010 8:21:54 by Hibernate Tools 3.2.1.GA -->
<hibernate-mapping>
  <class catalog="easyorm" name="hibernate.Banka" table="banka">
    <id name="bankaId" type="java.lang.Integer">
      <column name="banka_id"/>
      <generator class="identity"/>
    </id>
    <property name="nazev" type="string">
      <column length="50" name="nazev" not-null="true"/>
    </property>
    <property name="kod" type="string">
      <column length="4" name="kod" not-null="true"/>
    </property>
    <set inverse="true" name="ucets">
      <key>
        <column name="banka" not-null="true"/>
      </key>
      <one-to-many class="hibernate.Ucet"/>
    </set>
  </class>
</hibernate-mapping>
```

Výpis 14: XML – definice mapování pomocí externího souboru

4.4.2 Anotace

Anotace jsou doplňkové metadata, které rozšiřují možnosti definice tříd. Pomocí těchto metadat můžeme jednoduše definovat ORM přímo uvnitř zdrojového kódu perzistentní třídy. Díky tomu již nepotřebujeme externí mapovací soubor. Ve výpisu 15 je uveden příklad použití anotací.

```
@Entity
@Table(name = "banka")
@NamedQueries(
{
    @NamedQuery(name = "Banka.findAll", query = "SELECT b FROM Banka b")
})
public class Banka implements Serializable
{
    private static final long serialVersionUID = 1L;
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "banka_id", nullable = false)
    private Integer bankaId;
    @Basic(optional = false)
    @Column(name = "nazev", nullable = false, length = 50)
    private String nazev;
    @Basic(optional = false)
    @Column(name = "kod", nullable = false, length = 4)
    private String kod;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "banka")
    private List<Ucet> ucetList;5.2.3 Definice pomocí SQL skriptu

     .....
     .....
}
```

Výpis 15: Java – definice mapování pomocí anotací

4.4.3 Definice pomocí DDL

V tomto případě je ORM generováno na základě existujícího databázového schématu. Databázové schéma je definováno pomocí DDL příkazu jazyka SQL. Na základě tohoto schématu jsou vygenerovány jednotlivé perzistentní třídy. Ve výpisu 16 je uveden příklad databázového schématu.

```
CREATE TABLE Banka
(
   banka_id          INT NOT NULL PRIMARY KEY auto_increment,
   nazev             VARCHAR(50) NOT NULL,
   kod               VARCHAR(4) NOT NULL
);
```

Výpis 16: SQL skript – definice mapování pomocí DDL

Na výpisu 17 je perzistentní třída vygenerována podle tohoto schématu.

```
public class Uzivatel
{
    private int  uzivatel_id ;
    private String jmeno;
    private String prijmeni;
    private String email;
    private List<Forum> fora = new ArrayList<Forum>();

     .....
     .....
}

public class Forum
{
    private int forum_id;
    private String nazev;
```

```
    private List<Uzivatel> uzivatele = new ArrayList<Uzivatel>();

     .....
     .....
}
```

Výpis 17: Java – definice mapování pomocí DDL

5 ORM nástroje a frameworky

Tato kapitola se věnuje ORM nástrojům a frameworkům využívaným při tvorbě informačních systémů. Popisuje důvody jejich vzniku a zabývá se jejich efektivitou. Prezentuje také dva vybrané frameworky. Jedná se o ORM frameworky Hibernate a EclipseLink. Obsahuje zhodnocení použití ORM frameworků a upozorňuje na možná úskalí při jejich implementaci.

5.1 Teorie

V současné době se při vývoji většiny informačních systémů, využívá některý z objektově orientovaných programovacích jazyků. Tyto informační systémy, pak nejčastěji používají pro ukládání informací některý z relačních SŘBD. Mezi objektovým a relačním modelem jsou však velké rozdíly. Proto se při vytváření těchto IS věnuje velké množství času na zajištění ORM. Aby se nemuselo psát ručně, vznikly různé ORM nástroje a frameworky, které automaticky zajišťují potřebné ORM. Jejich hlavní úlohou je oddělení aplikační logiky od datové vrstvy informačního systému. Při použití některého z ORM nástrojů nebo frameworků, tedy již nemusí programátor psát zdrojový kód zajišťující objektově-relační mapování. Což podstatně sníží dobu potřebnou pro vytvoření informačního systému. Programátor ale ztrácí plnou kontrolu nad generovanými SQL příkazy.

Výhody vlastního ORM:

- plná kontrola nad generovanými SQL příkazy
- vlastní optimalizace pro konkrétní SŘBD

Nevýhody vlastního ORM:

- doba strávená implementací ORM
- většinou se neimplementují všechny funkcionality
- problematická změna používaného SŘBD

Výhody použití ORM nástrojů a frameworků:

- rychlejší tvorba aplikací
- jednoduchá změna používaného SŘBD

Nevýhody použití ORM nástrojů a frameworků:

- pouze částečná kontrola nad generovanými SQL příkazy
- nižší výkon

5.2 Efektivita mapování

Slabou stránkou ORM nástrojů a frameworků je částečné snížení výkonu aplikace při jejich použití. Sami o sobě zabírají určitou část procesorového času. Proto je žádoucí, aby byli co nejefektivnější. Zde jsou uvedeny nejdůležitější vlastnosti, které ovlivňují jejich efektivitu.

5.2.1 Líné načítání

Načítané objekty se mohou skládat z mnoha atributů různých velikostí. Také mohou odkazovat na další objekty, které mohou také odkazovat na další objekty atd. Při načtení takového objektu pak dochází k načtení všech jeho atributů a odkazovaných objektů. Někdy ale potřebujeme načíst jenom některé atributy objektu a je zbytečné načítat všechny atributy. Také ne vždy je nutné, načítat všechny odkazované objekty. Proto je výhodné, když ORM nástroj nebo framework podporuje tyto možnosti [22, 23]:

- načtení jednoho atributu
- načítání jen některých atributů
- načítání objektů jen do určité hloubky hierarchie

5.2.2 Ukládání jen změněných atributů

Již jsme viděli, že objekty mohou být velice rozsáhlé a mít složitou hierarchii. V případě že je modifikován takový objekt, může trvat dlouho, než se všechny jeho atributy a odkazované objekty uloží do databáze. Proto je vhodné, aby ORM nástroj nebo framework uměl ukládat pouze změněné atributy a objekty.

5.2.3 Explicitní zamykání

V případě že načítaný objekt bude modifikován, je vhodné ho načíst pomocí SQL dotazu SELECT FOR UPDATE. Tím se docílí toho, že nemůže dojít k souběhu v případě víceuživatelského přístupu k databázi. Toho je možné docílit také vhodným nastavením izolace transakcí.

5.2.4 Transakce

Transakce je skupina příkazů, které převádí databázi z jednoho konzistentního stavu do druhého. Při současném zpracovávání transakcí může dojít ke ztrátě konzistence a to, i když každá z těchto transakcí konzistenci zachovává. V případě víceuživatelského přístupu k databázi proto může docházet k těmto fenoménům:

- DIRTY READ – transakce čte data, které ještě jiná transakce nepotvrdila
- NONREPEATABLE READ – transakce znovu načte data a zobrazí jiné výsledky

- PHANTOM READ – transakce znovu vykoná dotaz vracející záznamy odpovídající určité podmínce a najde nové záznamy

V ANSI SQL jsou popsány tyto fenomény a definovány úrovně izolace transakcí, které tyto fenomény odstraňují. Úrovně izolace transakcí jsou tyto:

- READ UNCOMMITTED – může nastat DIRTY READ, NONREPEATABLE READ, PHANTOM READ
- READ COMMITTED – může nastat NONREPEATABLE READ, PHANTOM READ
- REPEATABLE READ – může nastat PHANTOM READ
- SERIALIZABLE - nemůže nastat žádný z fenoménů

Nejbezpečnější je úroveň SERIALIZABLE, při které se transakce provádějí za sebou a ne souběžně. Při jejím použití, ale může dojít ke snížení výkonu SŘBD. Také je nutné počítat s možnosti opakování transakce v případě zamítnutí přístupu. ORM nástroj nebo framework by měl umět pracovat s různými úrovněmi izolace transakcí.

5.2.5 Specifické vlastnosti SŘBD

Každý SŘBD má své specifické vlastnosti a metody, které umožňují dosáhnou nejlepšího výkonu tohoto databázového systému. Typickým příkladem jsou hromadné operace nad velkým množstvím záznamů. Pokud bude framework vyladěn na provádění hromadných operací pro konkrétní SŘBD, bude tyto operace provádět mnohem rychleji. Je tedy možné dosáhnou vyššího výkonu ORM frameworku tím, že do něj zabudujeme podporu specifických vlastností a metod pro různé druhy SŘBD.

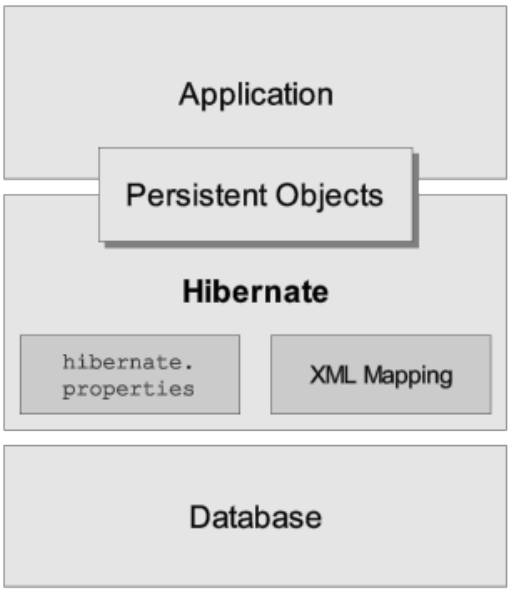
5.3 Hibernate

V současné době je to asi jeden z nejoblíbenějších frameworků pro objektově-relační mapování. Jedná se open source software poskytovaný pod licencí LGPL [17, 18]. K tomuto frameworku je k dispozici velice podrobná oficiální dokumentace. Navíc o něm bylo napsáno spoustu kvalitních publikací, podrobně rozebírajících všechny možné aspekty jeho použití.

5.3.1 Architektura

Framework Hibernate se používá v různých aplikačních architekturách. Lze jej použít jak v klasické Java SE(J2SE) aplikaci, tak i v aplikacích využívajících Java EE(J2EE) [17, 18]. Na obrázku 5 je vidět základní architektura tohoto frameworku.

Hibernate je konfigurován pomocí souboru $hibernate.properties$ a využívá XML soubory, které definují mapování tříd na tabulky v relační databázi [17]. Využívá JDBC pro komunikaci s touto databází [17]. To mu umožňuje jednoduše změnit používaný SŘBD.



Obrázek 5: Framework Hibernate – architektůra [17]

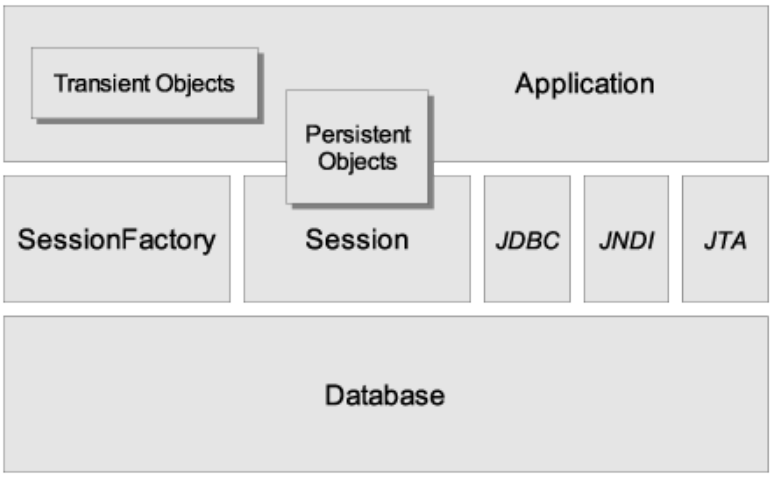
Framework nabízí několik variant použití, což umožňuje vybrat vhodnou variantu pro každý specifický projekt. Zde jsou prezentovány dvě extrémní varianty jeho použití. Na obrázku 6 je zobrazena varianta minimálního využití tohoto frameworku [17].

V tomto případě je využita jenom ta část Hibernate API, která zajišťuje JDBC připojení k SŘBD a správu transakcí. Na obrázku 7 je zobrazen druhý extrém, maximální využití tohoto frameworku [17].

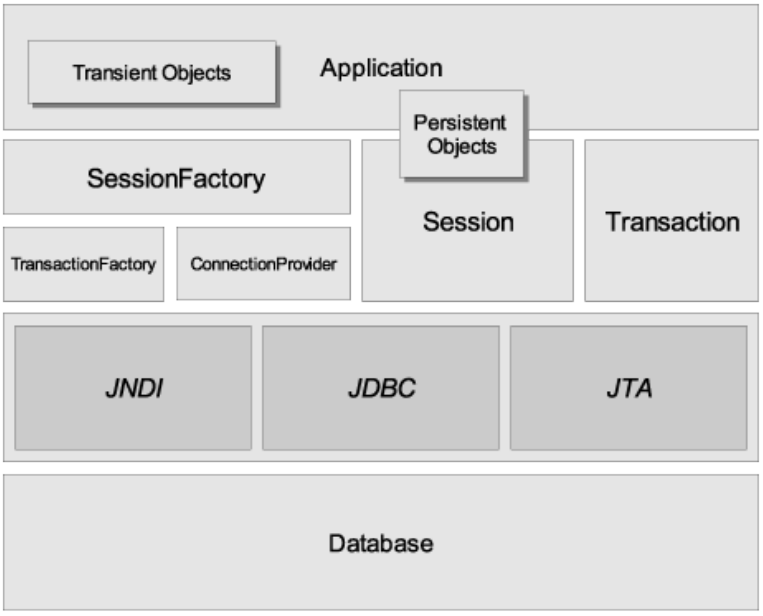
V tomto případě framework abstrahuje aplikaci od vrstev JDBC a JTA a sám se postará o vše potřebné.

Popis objektů [17]:

- $SessionFactory$ – instance třídy $SessionFactory$ obsahuje konfiguraci ORM a vytváří instance třídy $Session$
- $Session$ – instance třídy $Session$ zapouzdřuje JDBC spojení a provádí konverzi mezi objektovou a relační reprezentací dat, vytváří transakce a obsahuje první úroveň cache
- $PersistentObjects$ – instance perzistentních tříd, které jsou asociovány s aktuální $Session$. Po jejím ukončení, mohou být dále používány
- $TransientObjects$ – instance perzistentních tříd, které nejsou asociovány s aktuální $Session$
- $Transaction$ – instance třídy $Transaction$ zajišťují správný průběh operací s persistentními objekty, u kterých musí být zaručeno, že buď proběhnou všechny anebo neproběhnou vůbec



Obrázek 6: Frameworku Hibernate – minimální využití [17]



Obrázek 7: Frameworku Hibernate – maximální využití [17]

- $ConnectionProvider$ – instance třídy $ConnectionProvider$ vytváří JDBC spojení
- $TransactionFactory$ – továrna na instance třídy $Transaction$, není vytvářena, ale může být implementována programátorem

5.3.2 Vývojové scénáře

Framework Hibernate používá tyto vývojové scénáře [17, 18].

Top down

- vychází se z existujícího objektového modelu
- schéma databáze je vytvořeno pomocí mapovacího souboru nebo anotace
- vytváření databázového schématu pomocí utility hbm2ddl

Bottom up

- vychází se z existujícího databázového schématu
- utility na vygenerování mapovacího souboru, nebo anotací
- pomocí utility hbm2java je možné vygenerovat kostru zdrojového Java kódu

Middle out

- vychází se z existujícího mapovacího souboru ve formátu XML
- pomocí utility hbm2java je možné vygenerovat kostru zdrojového Java kódu
- vytváření databázového schématu pomocí utility hbm2ddl

Meet in the middle

- vychází se z existujícího databázového schématu a existujících Java tříd
- je nutné ručně napsat mapovací soubor mezi databázovým schématem a Java třídami (pravděpodobně bude také nutná úprava databázového schématu, nebo Java tříd)

5.4 Eclipse Link

Jedná se o open source framework pro ORM vyvinutý společností Eclipse Foundation [19]. Tento framework vychází z produktu TopLink, který vyvinula společnost Oracle [19, 21]. Dokáže používat libovolnou databázi, která má svůj JDBC ovladač. Dá se použít ve všech vývojových prostředích Javy a spolupracuje s různými aplikačními servery. Definice datového modelu může být umístěna v externím XML souboru, anebo může být zadána pomocí anotací. Na základě této definice automaticky vytvoří databázové schéma. Využívá technologii cachování, která uchovává data získaná z databáze. Jeho velkou výhodou je podrobná dokumentace a snadné použití.

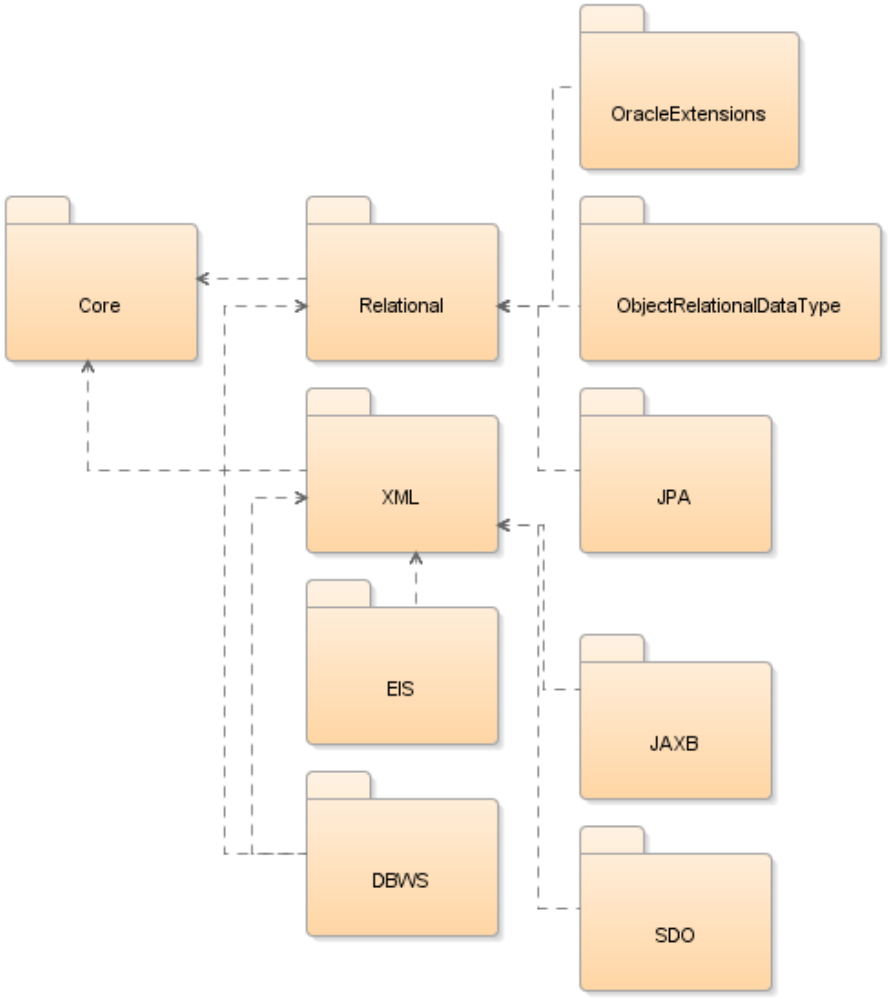
5.4.1 Architektura

Na obrázku 8 je zobrazen diagram komponent frameworku EclipseLink.
Framework se skládá z deseti komponent. Každá z nich plní určitou funkci:

- Core – objektové mapování a přístup k datům
- Relational – relační mapování a přístup k databázi pomocí JDBC
- OR data-type – mapování datových typů
- Oracle – rozšíření pro databázi Oracle
- EIS – přístup k EIS a šifrování
- XML – mapování a přístup ke XML souborům
- JAXB – Java API for XML
- JPA – Java Persistence API
- SDO – servisní služby
- DBWS – webové služby

5.5 Zhodnocení

ORM nástroje a frameworky mohou velice usnadnit vývoj informačních systémů. V některých případech ale mohou způsobovat problémy s výkonem. Proto je nutné, se s nimi velice dobře seznámit. Pak je možné většinu těchto problémů určitým způsobem vyřešit. At' už pomocí nastavení těchto nástrojů a frameworků, anebo přímým zadáním SQL dotazu.



Obrázek 8: Framework EclipseLink – diagram komponent [20]

6 Analýza a návrh

V této kapitole je popsána analýza a návrh ORM frameworku EasyORM. Jsou zde uvedeny jednotlivé požadavky na funkčnost frameworku a podrobná analýza těchto požadavků. Je zde proveden návrh vytvářeného frameworku a jsou zobrazeny jeho modely.

6.1 Požadavky na framework

Cílem této práce byla implementace objektově-relačního mapování pro platformu Java. Po nastudování různých technik objektově-relačního mapování, bylo rozhodnuto o návrhu a implementaci vlastního frameworku realizujícího toto mapování. Framework byl pojmenován EasyORM a při jeho implementaci byly použity nejnovější dostupné technologie.

6.1.1 Analýza požadavků

Framework by měl načítat DDL SQL skript ze souboru. Na jeho základě vytvořit třídy reprezentující entitní typy a třídy pro práci s databázovými tabulkami. Měl by také obsahovat pomocné třídy pro přístup k SŘBD, řízení transakcí a validaci instancí entitních tříd.

6.1.2 Technické požadavky

Technické požadavky na vytvářený framework jsou tyto:

- spolehlivost a stabilita - kvalitní a spolehlivé technologie
- aktuálnost - použití nejnovějších technologií
- platformovou nezávislost

6.1.3 Požadavky na funkčnost

Funkční požadavky na vytvářený framework jsou tyto:

- načtení DDL SQL skriptu
- generování tříd reprezentujících entitní typy
- generování tříd pro práci s databázovými tabulkami
- řízení transakcí
- poskytování poolingu pro připojení k databázi
- validace instancí entitních tříd

6.1.4 Uživatelské požadavky

Uživatelské požadavky na vytvářený framework jsou tyto:

- jednoduché použití
- vysoký výkon
- kvalitní dokumentace

6.2 Návrh řešení

Na základě analýzy specifikovaných požadavků byl vytvořen tento návrh řešení.

6.2.1 Dekompozice frameworku

Navrhovaný framework se skládá ze tří hlavních částí. První část obsahuje třídy, jejichž úkolem je:

- načtení DDL SQL skriptu
- generování tříd reprezentujících entitní typy
- generování tříd pro práci s databázovými tabulkami

Druhá část obsahuje pomocné třídy, jejichž úkolem je:

- řízení transakcí
- poskytování poolingu pro připojení k databázi
- validace instancí entitních tříd

Třetí část obsahuje vygenerované třídy.

6.2.2 Moduly frameworku

Kromě rozdělení na hlavní části můžeme tento framework rozdělit na tyto logické moduly.

Generování tříd Používá se ke generování tříd.

Validace Používá se k validaci entitních tříd.

Transakce Používá s k řízení transakcí.

SQL dotazy Slouží k realizaci SQL dotazů.

Načtení konfigurace Používá se k načtení konfigurace a předávání konfiguračních parametrů.

Připojení k databázi Administruje připojení k databázi a pooling.

6.2.3 Role v systému

V tomto frameworku se vyskytují tři různé role.

Programátor Toto je základní role v systému. Má přístup ke všem jeho modulům.

Entity V této roli vystupují vygenerované třídy reprezentující entitní typy. Má přístup pouze k modulu „Validace“.

EntityDB V této roli vystupují vygenerované třídy pro práci s databázovými tabulkami. Má přístup pouze k modulu „Připojení k databázi“.

6.3 Podrobná analýza frameworku

Na základě požadavků byl analyzován problém a navrženo jeho optimální řešení. Proto byly definovány vstupy a výstupy systému, dále byl navržen jeho funkční a datový model. Tyto údaje pak byly základem pro naprogramování frameworku.

6.3.1 Vstupy

Vstupy můžeme také označovat jako požadavky na zdroje frameworku. Vstupy umožňují vkládat data do frameworku. V navrhovaném frameworku budou dva vstupy.

6.3.1.1 Soubor „config.xml“ V tomto souboru je popsána konfigurace frameworku. Jsou v něm definovány následující parametry:

- JDBC ovladač
- URL k databázi
- počet vytvářených připojení k databázi
- účet k databázi
- heslo k databázi
- adresář pro ukládání generovaných tříd

6.3.1.2 Soubor s DDL SQL skriptem

Vstupní soubor s SQL příkazy pro vytvoření tabulek v databázi. Na základě tohoto souboru budou vygenerovány třídy reprezentující entitní typy a třídy pro práci s databázovými tabulkami.

6.3.2 Výstupy

Definováním výstupů jsou upřesněny požadavky na cílové chování frameworku. Je upřesněno, jaké výstupy budou vytvářeny a jaké vstupní data k tomu budou zapotřebí. Navrhovaný framework bude mít dva výstupy.

6.3.2.1 Třídy reprezentující entitní typy

Tyto třídy budou generovány na základě DDL příkazů na vytvoření tabulky v databázi. Vygenerované třídy budou obsahovat toto:

- stejné atributy jako tyto tabulky – typ atributů bude určen vhodnou konverzí, podle typu atributu databázové tabulky
- getter a setter metody pro atributy
- konstruktory
- metodu na vytištění atributů
- metodu pro validaci
- metodu pro uložení
- podporu pro líné načítání

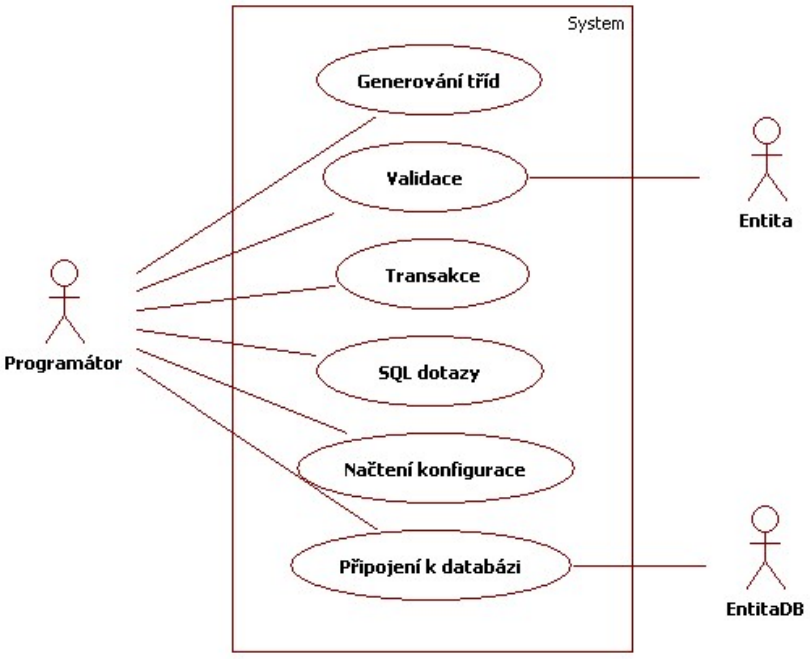
6.3.2.2 Třídy pro práci s databázovými tabulkami

Tyto třídy budou generovány na základě DDL příkazů na vytvoření tabulky v databázi. Vygenerované třídy budou obsahovat tyto metody pro práci s třídami reprezentujícími entitní typy:

- metodu pro uložení do databáze
- metody pro odstranění z databáze
- metody pro update
- metody pro načtení z databáze

6.3.3 Funkční specifikace

Funkční specifikace je zobrazena na obrázku 9 pomocí diagramu případu užití.



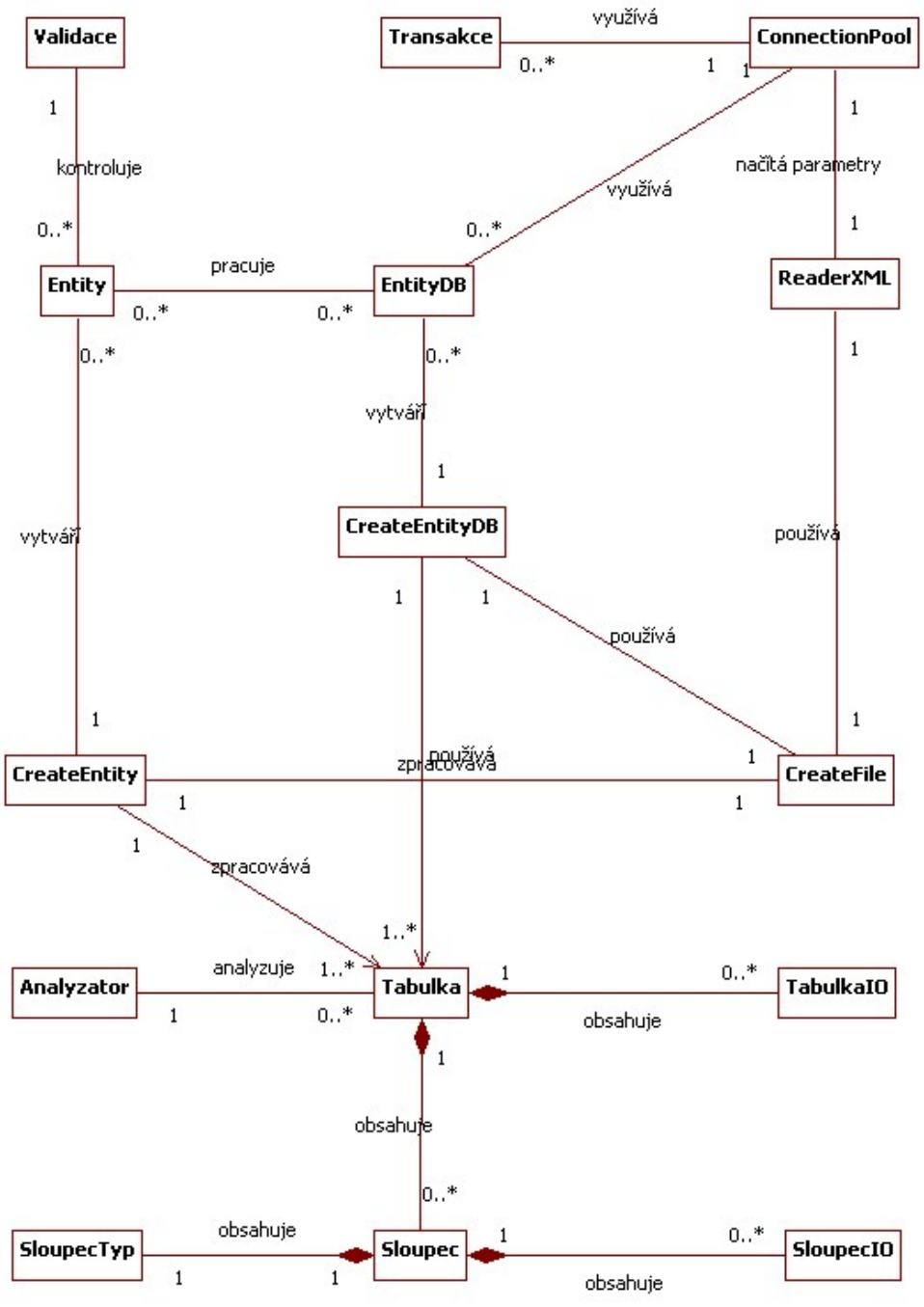
Obrázek 9: Diagram případů užití

6.3.4 Logická struktura systému

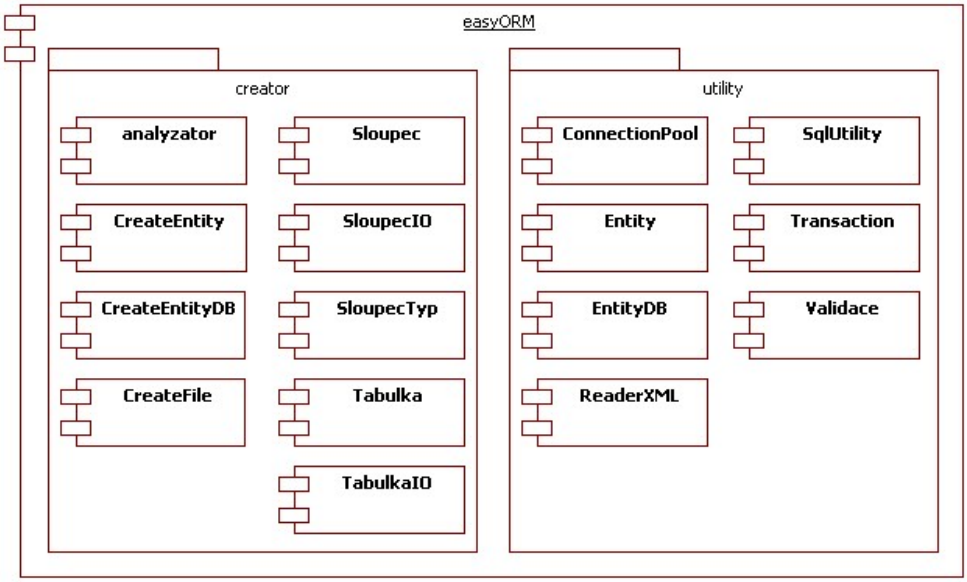
Na obrázku 10 je popsaná logická struktura frameworku pomocí jednoduchého diagramu tříd. Podrobný diagram tříd obsahující atributy a metody je v příloze A.

6.3.5 Specifikace implementace

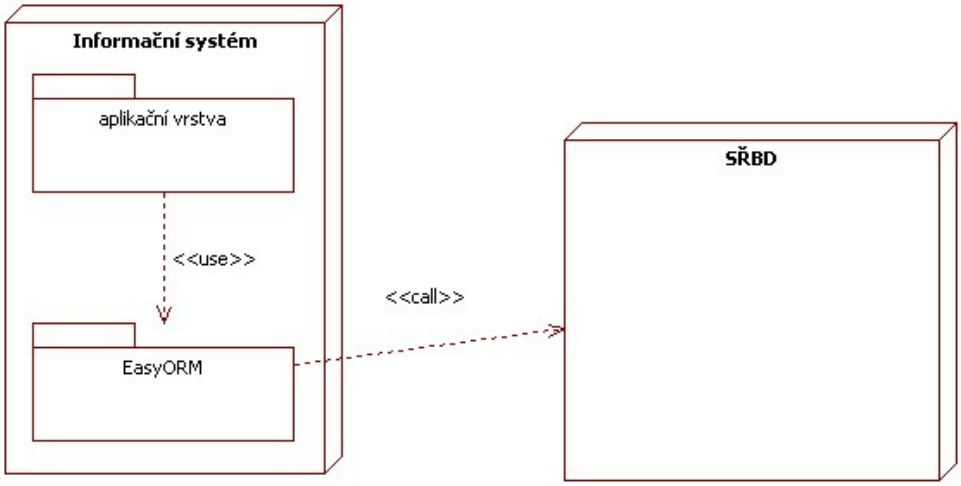
Specifikace implementace je definována pomocí diagramu komponent a diagramu nasazení. Na obrázku 11 je zobrazen diagram komponent a na obrázku 12 diagram nasazení.



Obrázek 10: Diagram tříd



Obrázek 11: Diagram komponent



Obrázek 12: Diagram nasazení

7 Implementace

V této kapitole je popsána implementace ORM frameworku EasyORM. Probírá se v ní výběr technologických prostředků a popisuje způsob implementace.

7.1 Výběr technologických prostředků

V dnešní době se pro vývoj informačních systému dá použít více objektově orientovaných programovacích jazyků. Může se použít například Java, C++, C#, Visual Basic, dokonce se často používá i PHP. Pro tuto práci ale byla určena platforma Java. V současné době existují dvě hlavní vývojová prostředí pro tuto platformu. Jsou to vývojová prostředí Eclipse a Netbeans. Obě splňují požadavky na vývoj tohoto frameworku. Nakonec bylo vybráno vývojové prostředí Netbeans. Navrhované řešení tedy bylo realizováno pomocí programovacího jazyka Java v kombinaci s vývojovým prostředím NetBeans.

7.1.1 Java

Java je objektově orientovaný programovací jazyk, vyvinutý firmou Sun Microsystems. Java patří mezi nejpoužívanější programovacích jazyky ve světě. Pro svou vynikající přenositelnost je používán pro tvorbu aplikací pracujících na různých operačních systémech. Používá se v různých dílčích platformách [24].

JavaCard Platforma pro vývoj aplikací provozovaných na čipových kartách.

Java ME Platforma pro vývoj aplikací provozovaných na mobilních zařízeních.

Java SE Platforma pro vývoj aplikací provozovaných na desktopových počítačích.

Java EE Platforma pro vývoj rozsáhlých distribuovaných systémů.

Souhrn těchto platforem se nazývá platforma Java. Od května 2007 je vyvíjena jako Open Source projekt [24].

7.1.2 Netbeans IDE

Vývojové prostředí NetBeans IDE je nástroj, pomocí kterého programátoři mohou psát, překládat, ladit a distribuovat aplikace [25]. Samotné vývojové prostředí je vytvářeno v jazyce Java - ovšem podporuje prakticky jakýkoliv programovací jazyk. Existuje rovněž velké množství modulů, které toto vývojové prostředí rozšiřují. Vývojové prostředí NetBeans je bezplatně šířený produkt a jeho užívání není nijak omezeno.

7.1.3 Kódování

Pro správné zobrazování českých znaků je třeba zvolit vhodné kódování. Vybral jsem kódování UTF-8, které se dnes pro kódování textu ve znakové sadě Unicode používá asi nejčastěji. Zvláštností UTF-8 je to, že jeden znak může zakódovat do proměnlivého počtu bytů (jednoho až čtyř). Prvních 128 znaků Unicode je převzato z ASCII. UTF-8 znaky s kódem menším než 128 přímo kóduje jako jeden byte. Znaky s kódy většími než 128 jsou reprezentovány jako několik bytů.

7.2 Specifikace použitých technologií

Pro implementaci frameworku jsem použil tyto technologie.

PC:

- Intel Core 2 Quad Q6600 2,4GHz
- RAM 3000MB DDR3
- HDD 150GB SATAII, 10k
- MS Windows XP SP3
- NetBeans IDE 6.8

IDE:

- NetBeans IDE 6.8
- Java 6 SE

7.3 Balíčky

Implementovaný framework se skládá ze tří balíčků. První balíček se nazývá „creator“, druhy „utility“ a třetí „create“.

7.3.1 Balíček „creator“

Tento balíček slouží k vytváření tříd reprezentujících entitní typy a tříd pro práci s databázovými tabulkami. Součásti tohoto balíčku jsou následující třídy.

Analyzátor Tato třída načte konfigurační parametry a otevře soubor s DDL SQL skriptem. Provádí podrobnou analýzu tohoto skriptu. Na základě této analýzy vytváří instance třídy tabulka.

CreateEntity Na základě instance třídy tabulka vygeneruje entitní třídu. Třída je uložena do adresáře, definovaného v konfiguračních parametrech. Vygenerovaná třída obsahuje všechny potřebné metody definované ve výstupech.

CreateEntityDB Na základě instance třídy tabulka vygeneruje třídu pro práci s touto databázovou tabulkou. Třída je uložena do adresáře, definovaného v konfiguračních parametrech. Vygenerovaná třída obsahuje všechny potřebné metody definované ve výstupech.

CreateFile Tato třída se používá pro vytvoření souboru na disku. Soubor je uložen do adresáře, definovaného v konfiguračních parametrech.

Sloupec Tato třída se používá pro reprezentaci sloupce v databázové tabulce. Obsahuje metodu pro výpis atributů sloupce.

SloupecIO Tato třída se používá pro reprezentaci integritního omezení sloupce v databázové tabulce. Obsahuje výčtový typ IOSloupce, který reprezentuje integritní omezení sloupců podle standardu ANSI SQL.

SloupecTyp Tato třída se používá pro reprezentaci typu sloupce v databázové tabulce. Obsahuje výčtový typ TypSloupce, který reprezentuje typy sloupců podle standardu ANSI SQL. Obsahuje metodu na převod těchto typů na datové typy v Javě.

Tabulka Tato třída se používá pro reprezentaci databázové tabulky. Obsahuje metody pro výpis atributů tabulky, nalezení primárního klíče a zjišťování existence datových typů.

TabulkaIO Tato třída se používá pro reprezentaci integritního omezení databázové tabulky. Obsahuje výčtový typ IOTabulky, který reprezentuje integritní omezení tabulky podle standardu ANSI SQL.

7.3.2 Balíček „utility“

Tento balíček slouží k poskytování podpory pro vygenerované třídy. Součásti tohoto balíčku jsou následující třídy.

ConnectionPool Tato třída vytváří na základě konfiguračních parametrů požadovaný počet připojení k databázi. Pro zvýšení výkonu frameworku poskytuje pooling těchto připojení. Je vytvářená podle návrhového vzoru Singleton.

Entity Abstraktní předek pro vygenerované třídy reprezentující entitní typy.

EntityDB Abstraktní předek vygenerovaných tříd pro práci s databázovými tabulkami.

ReaderXML Tato třída načítá konfigurační XML soubor a předává dalším třídám načtené konfigurační parametry. Je vytvářená podle návrhového vzoru Singleton.

SqlUtility Tato třída provádí zadaný SQL dotaz a vrací získaný resultát.

Transaction Tato třída poskytuje metody pro práci s transakcemi.

Validace Tato třída poskytuje metody pro validaci entitních tříd.

7.3.3 Balíček „create“

Do tohoto balíčku jsou standardně ukládány vygenerované třídy. Toto jde změnit pomocí konfiguračního souboru.

7.3.4 Konfigurační soubor

Konfigurační soubor byl implementován pomocí XML souboru ”config.xml”. V tomto souboru jsou definovány všechny požadované parametry. Ve výpisu 18 je uveden možný tvar tohoto souboru.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Document : config.xml
    Created on : 31. prosinec 2009, 8:07
    Author     :
    Description:
-->
<root>
    <!-- Konfigurace připojení k databázi databázi -->
    <databaze>
        <!-- JDBC ovladač poskytovány výrobcem databáze -->
        <ovladac>com.mysql.jdbc.Driver</ovladac>

        <!-- URL k databázi -->
        <url>jdbc:mysql://localhost:3306/EasyORM</url>
        <!-- <url>jdbc:derby://localhost:1527/sample</url> -->

        <!-- Počet vytvořených připojení k databázi -->
        <pocet>10</pocet>

        <!-- Účet k databázi -->
        <login>root</login>

        <!-- Heslo k databázi -->
        <password>211</password>

        <!-- Adresář pro ukládání vytvářených souborů -->
        <adresar>./src/create/</adresar>
    </databaze>
</root>
```

Výpis 18: Konfigurační soubor „config.xml“

8 Porovnání frameworků ORM

V této kapitole jsou definovány metody porovnávání vybraných ORM frameworků. Je zde popsán způsob měření výkonu jednotlivých ORM frameworků a jsou zde prezentovány výsledky porovnávání.

8.1 Metody porovnávání

V této podkapitole jsou definovány metody porovnávání vybraných ORM frameworků s implementovaným frameworkem. Je definován způsob měření výkonu jednotlivých ORM frameworků. K tomu účelu bylo vytvořeno několik testovacích dotazů, na kterých byla provedena tato měření.

8.1.1 Kritéria porovnávání

Pro porovnávání výkonu vytvořeného frameworku s dostupnými ORM frameworky, byl vytvořen jednoduchý objektový model. Skládá se ze sedmi tříd a modeluje evidenci převodů finančních částek v bankovním systému. Při generování datové vrstvy se, ale vycházelo z existujícího databázového schématu. Pro ukládání dat byly použity dva různé SŘBD. První byl zvolen MySQL 6.0, jako reprezentant open source SŘBD. Jako druhý byl vybrán MS SQL Server 2008, který je typickým představitelem komerčních SŘBD. Tabulky databáze byly naplněny náhodně vygenerovanými daty. Nad těmito daty byly pomocí testovacích dotazů provedeny standardní databázové operace Read a Update. Byla změřena doba vykonávání dotazů při využití jednotlivých frameworků a použití vybraných SŘBD. Vždy byla změřena doba vykonání jednoho dotazu, aby bylo možné eliminovat dobu inicializace frameworku. Pak byl dotaz vykonán dvacetkrát a byla určena průměrná doba vykonání dotazu.

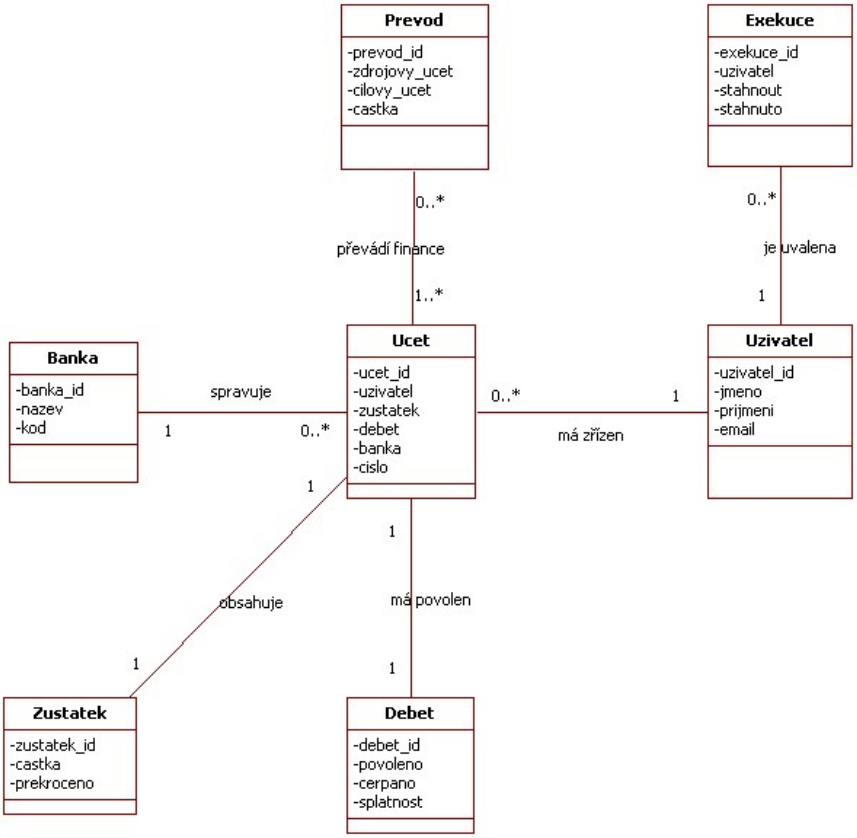
8.1.1.1 Specifikace použitých technologií Pro testování výkonu jednotlivých frameworků byly použity tyto technologie.

PC:

- Intel Core 2 Quad Q6600 2,4GHz
- RAM 3000MB DDR3
- HDD 150GB SATAII, 10k
- MS Windows XP SP3
- NetBeans IDE 6.8

SŘBD:

- MySQL 6.0
- MS SQL Server 2008



Obrázek 13: Diagram tříd - testovaný model

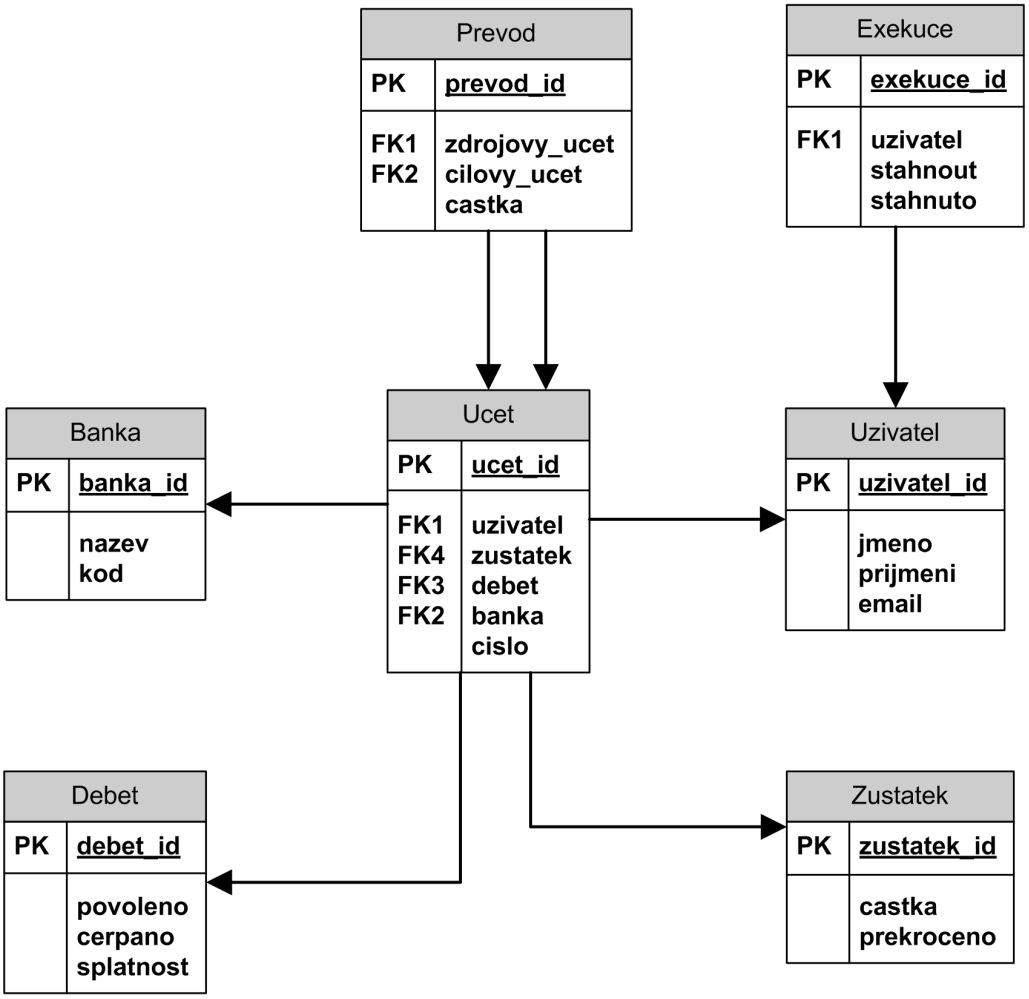
ORM frameworky:

- EasyORM
- Hibernate
- EclipseLink

8.1.1.2 Diagram tříd Na obrázku 13 je diagram tříd reprezentuje jednoduchou aplikaci na evidenci finančních převodů mezi jednotlivými uživateli.

8.1.1.3 ER Diagram Na obrázku 14 je ER diagram reprezentuje konceptuální znázornění dat v relační databázi.

8.1.1.4 Vytvoření testovací databáze Pro vytvoření databáze byl použit tento SQL skript.



Obrázek 14: ER diagram - testovací databáze

```
CREATE TABLE Uzivatel
(
   CREATE DATABASE Test;
   USE Test;

CREATE TABLE Uzivatel
(
    uzivatel_id     INT NOT NULL PRIMARY KEY auto_increment,
   jmeno            VARCHAR(20) NOT NULL,
   prijmeni         VARCHAR(20) NOT NULL,
   email            VARCHAR(50) NOT NULL
);

CREATE TABLE Exekuce
(
   exekuce_id       INT NOT NULL PRIMARY KEY auto_increment,
    uzivatel        INT NOT NULL,
   stahnout         DECIMAL(14,2) NOT NULL,
   stahnuto         DECIMAL(14,2) NOT NULL
);

CREATE TABLE Banka
(
   banka_id         INT NOT NULL PRIMARY KEY auto_increment,
   nazev            VARCHAR(50) NOT NULL,
   kod              VARCHAR(4) NOT NULL
);

CREATE TABLE Zustatek
(
   zustatek_id      INT NOT NULL PRIMARY KEY auto_increment,
   castka           DECIMAL(14,2) NOT NULL,
   prekroceno       DATE
);

CREATE TABLE Debet
(
   debet_id         INT NOT NULL PRIMARY KEY auto_increment,
   povoleno         DECIMAL(14,2) NOT NULL,
   cerpano          DECIMAL(14,2) NOT NULL,
   splatnost            INT NOT NULL
);

CREATE TABLE Ucet
(
    ucet_id             INT NOT NULL PRIMARY KEY auto_increment,
    uzivatel            INT NOT NULL,
    zustatek            INT NOT NULL,
    debet               INT NOT NULL,
    banka               INT NOT NULL,
    cislo               VARCHAR(50) NOT NULL,
   FOREIGN KEY (uzivatel) REFERENCES Uzivatel (uzivatel_id),
   FOREIGN KEY (zustatek) REFERENCES Zustatek (zustatek_id),
```

```
    FOREIGN KEY (debet) REFERENCES Debet (debet_id),
    FOREIGN KEY (banka) REFERENCES Banka (banka_id)
);

CREATE TABLE Prevod
(
    prevod_id           INT NOT NULL PRIMARY KEY auto_increment,
    zdrojovy_ucet       INT NOT NULL,
    cilovy_ucet         INT NOT NULL,
    castka              DECIMAL(14,2) NOT NULL,
    FOREIGN KEY (zdrojovy_ucet) REFERENCES Ucet (ucet_id),
    FOREIGN KEY (cilovy_ucet) REFERENCES Ucet (ucet_id)
);

);
```

Výpis 19: SQL skript – vytvoření testovací databáze

8.1.1.5 Generování dat Pro účely testování jednotlivých frameworků byly vytvořeny tabulky, naplněny náhodně vygenerovanými záznamy.

Tabulka $Uzivatel$:

- vygenerováno 10000 záznamů
- atribut $uzivatel\_id$ - automaticky generován databází
- atribut $jmeno$ - náhodná hodnota z 20 nejčastějších mužských jmen
- atribut $prijmeni$ - náhodná hodnota z 20 nejčastějších mužských příjmení
- atribut $email$ - kombinace jména, příjmení a 5 náhodných domén

Tabulka $Exekuce$:

- vygenerováno 3000 záznamů
- atribut $exekuce\_id$ - automaticky generován databází
- atribut $uzivatel$ - náhodně vybrané existující peněžní ústavy
- atribut $stahnout$ - náhodná hodnota z intervalu 1 až 1000000
- atribut $stahnuto$ - náhodná hodnota z intervalu 1 až $stahnout$

Tabulka $Banka$:

- vygenerováno 10 záznamů
- atribut $banka\_id$ - automaticky generován databází
- atribut $jmeno$ a kod - náhodně vybrané existující peněžní ústavy

Tabulka $Zustatek$:

- vygenerováno 20000 záznamů

- atribut $zustatek\_id$ - automaticky generován databází
- atribut $castka$ - náhodná hodnota z intervalu 1 až 1000000
- atribut $prekroceno$ - náhodné datum

Tabulka $Debet$:

- vygenerováno 3000 záznamů
- atribut $debet\_id$ - automaticky generován databází
- atribut $povoleno$ - náhodná hodnota z intervalu 0 až 200000
- atribut $cerpano$ - náhodná hodnota z intervalu 0 až $povoleno$
- atribut $splatnost$ - náhodná hodnota z intervalu 30 až 100

Tabulka $Ucet$:

- vygenerováno 20000 záznamů
- atribut $ucet\_id$ - automaticky generován databází
- atribut $uzivatel$ - náhodná hodnota z intervalu 1 až 10000
- atribut $zustatek$ - náhodná hodnota z intervalu 1 až 20000
- atribut $debet$ - náhodná hodnota z intervalu 1 až 20000
- atribut $banka$ - náhodná hodnota z intervalu 1 až 10
- atribut $cislo$ - náhodná hodnota z intervalu 100M až 1000M

Tabulka $Prevod$:

- vygenerováno 1000000 záznamů
- atribut $prevod\_id$ - automaticky generován databází
- atribut $zdrojovy\_ucet$ - náhodná hodnota z intervalu 1 až 20000
- atribut $cilovy\_ucet$ - náhodná hodnota z intervalu 1 až 20000
- atribut $castka$ - náhodná hodnota z intervalu 1 až 1000000

8.1.2 Testovací framework

Pro vyhodnocení testování byl připraven jednoduchý testovací framework. Tento framework se skládá z šesti základních tříd:

- CreateDatabase - naplní testovací tabulky náhodně vygenerovanými daty
- EasyORMTester - slouží k testování výkonu ORM frameworku EasyORM
- EclipseLinkTester - slouží k testování výkonu ORM frameworku EclipseLink
- HibernateTester - slouží k testování výkonu ORM frameworku Hibernate
- Tester - vyhodnotí výsledky testů jednotlivých frameworků
- Timer - pomocná třída pro měření času

Pomocí tohoto frameworku bylo provedeno porovnání jednotlivých ORM frameworků.

8.1.2.1 Diagram tříd Na obrázku 15 je zobrazen diagram tříd, reprezentující tento jednoduchý testovací framework.

8.1.3 Testovací dotazy

Pro porovnání frameworků bylo vytvořeno několik testovacích dotazů. Byly vybírány takové dotazy, na kterých je možné ukázat nejčastější chyby při používání ORM frameworků.

Testovací dotaz č. 1 Jako první byl vybrán poměrně jednoduchý dotaz. V tabulce $Prevod$ se vyhledá celková suma převáděných částek. Algoritmus tohoto dotazu byl formulován takto:

- použít alternativu tohoto SQL dotazu - „SELECT * FROM prevod“
- pomocí ORM frameworku načíst kolekci všech objektů z databáze
- projít celou kolekci a spočítat součet převáděných částek

Tato varianta sice vypadá jako nejjednodušší, ale je časově a výkonově velmi náročná. Znamená to načíst celý milion záznamů z této tabulky do operační paměti, vytvořit kolekci s celým milionem objektů třídy $Prevod$, projít jednotlivé objekty kolekce a spočítat součet jejich atributu $castka$.

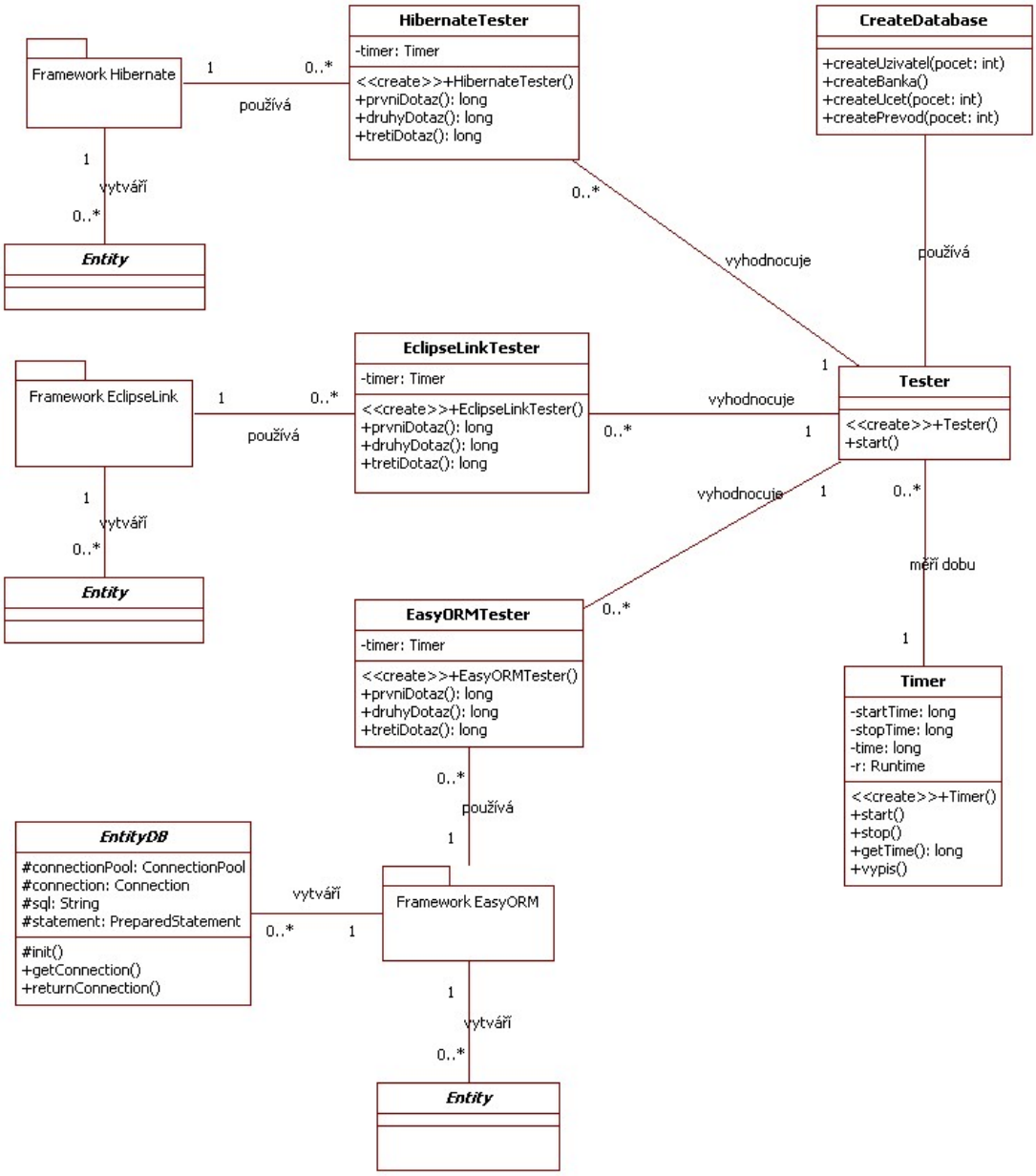
Testovací dotaz č. 2 Tento dotaz je stejný jako předchozí. V tabulce $Prevod$ se vyhledá celková suma převáděných částek. Algoritmus tohoto dotazu byl, ale formulován jinak:

- použít alternativu tohoto SQL dotazu - „SELECT SUM(castka) AS soucet FROM prevod“
- pomocí ORM frameworku přímo načteme hodnotu celkového součtu

V tomto případě je načtena z databáze pouze jediná hodnota reprezentující požadovaný součet. Předpokládáme, že tento postup je mnohonásobně rychlejší a spotřebuje podstatně méně systémových prostředků než u předchozího dotazu.

Testovací dotaz č. 3 Vyhledávají se informace o všech převodech, provedených z účtu konkrétního uživatele (např. Petra Nováka). Algoritmus tohoto dotazu byl formulován takto:

- použít alternativu tohoto SQL dotazu - „SELECT * FROM Ucet U, Uzivatel Uz, Prevod P WHERE Uz.jmeno='Petr' AND Uz.prijmeni = 'Novák' AND Uz.uzivatel_id = U.uzivatel_id AND U.ucet_id = P.ucet_id“
- pomocí ORM frameworku načíst kolekci všech objektů z databáze



Obrázek 15: Diagram tříd - testovací framework

- projít celou kolekci a vytisknout údaje o převodech

Tento způsob je velice jednoduchý, ale při jeho provádění se provede kartézský součin tří tabulek. Což znamená, že je nutné projít $1000 * 2000 * 1000000 = 2 * 10E12$ záznamů, aby byly zjištěny požadované informace.

Testovací dotaz č. 4 Tento dotaz je stejný jako předchozí. Vyhledávají se informace o všech převodech provedených z účtu konkrétního uživatele (např. Petra Nováka). Algoritmus tohoto dotazu byl ale formulován jinak:

- použít alternativu tohoto SQL dotazu - „SELECT * FROM Prevod WHERE zdrojov_ucet IN (SELECT ucet_id FROM Ucet WHERE uzivatel_id IN (SELECT uzivatel_id FROM Uzivatel WHERE jmeno='Petr' AND prijmeni = 'Novák'"
- pomocí ORM frameworku načíst kolekci všech objektů z databáze
- projít celou kolekci a vytisknout údaje o převodech

Uživatel se hledá v 10000 záznamech a výsledkem je jeden záznam. Ten se porovná z 20000 účty a výsledkem je X (možná také žádný nebo pouze jeden) záznamů o účtech uživatele. Ty se porovnají s 1000000 záznamů o převodech. Celkem tedy 10000 + 20000 + X*1000000 = 1030000 porovnání, pokud má uživatel pouze jeden účet.

Testovací dotaz č. 5 V tomto případě se bude vkládat do tabulky $Uzivatel$ 10000 nových uživatelů. Algoritmus tohoto dotazu byl formulován takto:

- začátek transakce
- vygenerování 10000 objektů třídy $Uzivatel$
- pomocí metody insert uložíme objekty do databáze
- konec transakce

Tímto způsobem otestujeme rychlost frameworků při operaci insert.

Testovací dotaz č. 6 Všem uživatelům, kteří mají exekuci na účtu, připiš k exekuci jednorázový poplatek ve výši 500,- Kč. Algoritmus tohoto dotazu byl formulován takto:

- začátek transakce
- pomocí ORM frameworku je načtena kolekce všech objektů třídy $Exekuce$ z databáze
- každému objektu přičteme k atributu $stahnout$ částku 500 Kč

- pomocí metody update(Zustatek zustatek) postupně uložíme objekty zpátky do databáze
- konec transakce

Tento způsob je relativně jednoduchý, ale při jeho provádění se ukládají do databáze všechny atributy objektu. Změněn byl, ale pouze jeden atribut. V případě že objekt má některé atributy většího rozsahu (např. obrázky), může dojít ke snížení výkonu frameworku. Na tomto dotazu. Na tomto dotazu bude ukázána častá chyba při práci se SŘBD. Touto chybou je nevkládání insertů a updatu do transakce.

Testovací dotaz č. 7 Tento dotaz je stejný jako dva předchozí. Všem uživatelům, kteří mají exekuci na účtu, připiš k exekuci jednorázový poplatek ve výši 500,- Kč. Algoritmus tohoto dotazu byl, ale formulován jinak:

- pomocí ORM frameworku je načtena kolekce všech objektů třídy $Exekuce$ z databáze
- každému objektu přičteme k atributu $stahnout$ částku 500 Kč
- začátek transakce
- pomocí metody update(Exekuce oldExekuce, Exekuce newExekuce) postupně uložíme objekty zpátky do databáze
- konec transakce

V tomto případě se do databáze ukládají pouze změněné atributy. Tento postup by měl být rychlejší, ale ne všechny ORM Frameworky jej podporují.

Testovací dotaz č. 8 Všem uživatelům, je nutné navýšit povolený debet o 10000 Kč. Algoritmus tohoto dotazu byl formulován takto:

- pomocí ORM frameworku je načtena kolekce všech objektů třídy $Exekuce$ z databáze
- každému objektu přičteme k atributu $stahnout$ částku 500 Kč
- začátek transakce
- pomocí metody update(Exekuce oldExekuce, Exekuce newExekuce) postupně uložíme objekty zpátky do databáze
- konec transakce

V tomto případě ale budeme testovat paralelní přístup k SŘBD. Změříme jaký bude výkon při dvou různých úrovních izolace transakcí. Použijeme úroveň READ_COMMITED a SERIALIZABLE. Budou spuštěny dva tyto dotazy současně a budeme testovat rychlost vykonání těchto dotazů.

Testovací dotaz č. 9 Zjistěte zůstatek na všech účtech uživatelů. Algoritmus tohoto dotazu byl formulován takto:

- použít alternativu tohoto SQL dotazu - „SELECT * FROM Zustatek“
- pomocí ORM frameworku načíst kolekci všech objektů z databáze
- projít celou kolekci a spočítat součet převáděných částek

V tomto případě ale budeme testovat využití cachování u jednotlivých frameworků. Pro lepší ukázku využití cachování byl navýšen počet záznamů v tabulce $Zustatek$ na 100000.

8.2 Výsledky porovnávání

V této podkapitole jsou prezentovány výsledky porovnávání ORM frameworků. U každého dotazu je rovněž zobrazen graf a tabulka s naměřenými hodnotami.

8.2.1 Dotaz č. 1

8.2.1.1 Implementace Tady jsou zobrazeny ukázky zdrojových kódů, pomocí kterých byl implementován tento dotaz. Jsou zobrazovány pouze určité části kódu, pro přehlednost je vypuštěno například ošetření výjimek.

EasyORM Ve výpisu kódu 20 je zobrazena implementace pro framework EasyORM.

```
 ......
double soucet = 0;
PrevodDB prevodDB = new PrevodDB();
List<Prevod> seznam = new ArrayList<Prevod>();

seznam = prevodDB.getAll();

for (Prevod element : seznam)
{
    soucet += element.getCastka();
}
 ......
```

Výpis 20: Java – implementace dotazu č. 1 na frameworku EasyORM

EclipseLInk Ve výpisu kódu 21 je zobrazena implementace pro framework EclipseLink.

```
 ......
double soucet = 0;
List<Prevod> seznam = new ArrayList<Prevod>();
EntityManagerFactory factory = Persistence.createEntityManagerFactory("Tester2PU");
EntityManager em = factory.createEntityManager();

em = factory.createEntityManager();
Query query = em.createQuery("SELECT p FROM Prevod p");
seznam = query.getResultList();

for (Prevod element : seznam)
{
  soucet += element.getCastka().doubleValue();
}
 ......
```

Výpis 21: Java – implementace dotazu č. 1 na frameworku EclipseLink

Framework	EasyORM		Hibernate		EclipseLink	
SŘBD	Inicial.	Průměr	Inicial.	Průměr	Inicial.	Průměr
MySQL 6.0	6344	4868	27406	23728	80922	79890
MS SQL Server 2008	8126	6752	28605	24912	83114	82726

Tabulka 1: Testovací dotaz č. 1

Hibernate Ve výpisu kódu 22 je zobrazena implementace pro framework Hibernate.

```
 ......
double soucet = 0;
Session session = null;
List<Prevod> seznam = new ArrayList<Prevod>();

session = HibernateUtil.getSessionFactory().openSession();
seznam = (List<Prevod>) session.createQuery("from Prevod").list();

for (Prevod element : seznam)
{
   soucet += element.getCastka().doubleValue();
}
 ......
```

Výpis 22: Java – implementace dotazu č. 1 na frameworku Hibernate

8.2.1.2 Výsledky a vyhodnocení V tabulce 1 jsou zobrazeny časy vykonávání tohoto dotazu u vybraných ORM frameworku. Naměřené hodnoty jsou uváděny v milisekundách. Na obrázku 16 je zobrazen sloupcový graf reprezentující naměřené hodnoty.

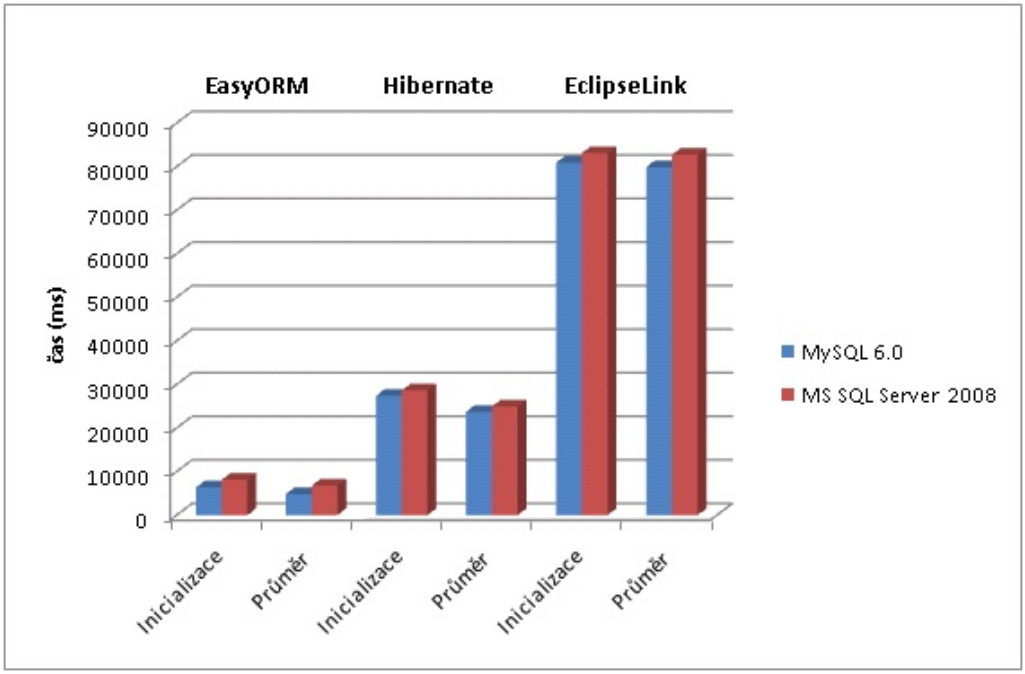
Na základě naměřených hodnot bylo určeno toto pořadí frameworků:

1. EasyORM
2. Hibernate
3. EclipseLink

Nejrychleji provedl tento dotaz framework EasyORM. Oba ostatní frameworky jak Hibernate, tak EclipseLink byly výrazně horší. Bylo to způsobeno velkou režii při vytváření objektového modelu pro tak velký počet objektů.

8.2.2 Dotaz č. 2

8.2.2.1 Implementace Tady jsou zobrazeny ukázky zdrojových kódů, pomocí kterých byl implementován tento dotaz. Jsou zobrazovány pouze určité části kódu, pro přehlednost je vypuštěno například ošetření výjimek.



Obrázek 16: Graf - testovací dotaz č. 1

EasyORM Ve výpisu kódu 23 je zobrazena implementace pro framework EasyORM.

```
 ......
double soucet = 0;
 SqlUtility    sqlUtility   = new SqlUtility () ;
ResultSet resultSet = null;
resultSet = sqlUtility .executeQuery("SELECT␣SUM(castka)␣AS␣soucet␣FROM␣Prevod");
soucet = resultSet.getDouble("soucet");
 ......
```

Výpis 23: Java – implementace dotazu č. 2 na frameworku EasyORM

EclipseLInk Ve výpisu kódu 24 je zobrazena implementace pro framework EclipseLink.

```
 ......
double soucet = 0;
 List<Prevod> seznam = new ArrayList<Prevod>();
EntityManagerFactory factory = Persistence.createEntityManagerFactory("Tester2PU");
EntityManager em = factory.createEntityManager();

Query query = em.createQuery("SELECT␣SUM(p.castka)␣AS␣soucet␣FROM␣Prevod␣p");
 List  result  = query.getResultList() ;

for (Object element : result)
{
    soucet = Double.valueOf(element.toString());
```

Framework	EasyORM		Hibernate		EclipseLink	
SŘBD	Inicial.	Průměr	Inicial.	Průměr	Inicial.	Průměr
MySQL 6.0	735	389	1766	390	1859	391
MS SQL Server 2008	842	453	1857	456	1976	459

Tabulka 2: Testovací dotaz č. 2

```
}
 ......
```

Výpis 24: Java – implementace dotazu č. 2 na frameworku EclipseLink

Hibernate Ve výpisu kódu 25 je zobrazena implementace pro framework Hibernate.

```
 ......
Session session = null;
double soucet = 0;
session = HibernateUtil.getSessionFactory().openSession();

Query q = session.createQuery("SELECT SUM (castka) AS soucet FROM Prevod");
List result = q.list();

for (Object element : result)
{
   soucet = Double.valueOf(element.toString());
}
 ......
```

Výpis 25: Java – implementace dotazu č. 2 na frameworku Hibernate

8.2.2.2 Výsledky a vyhodnocení

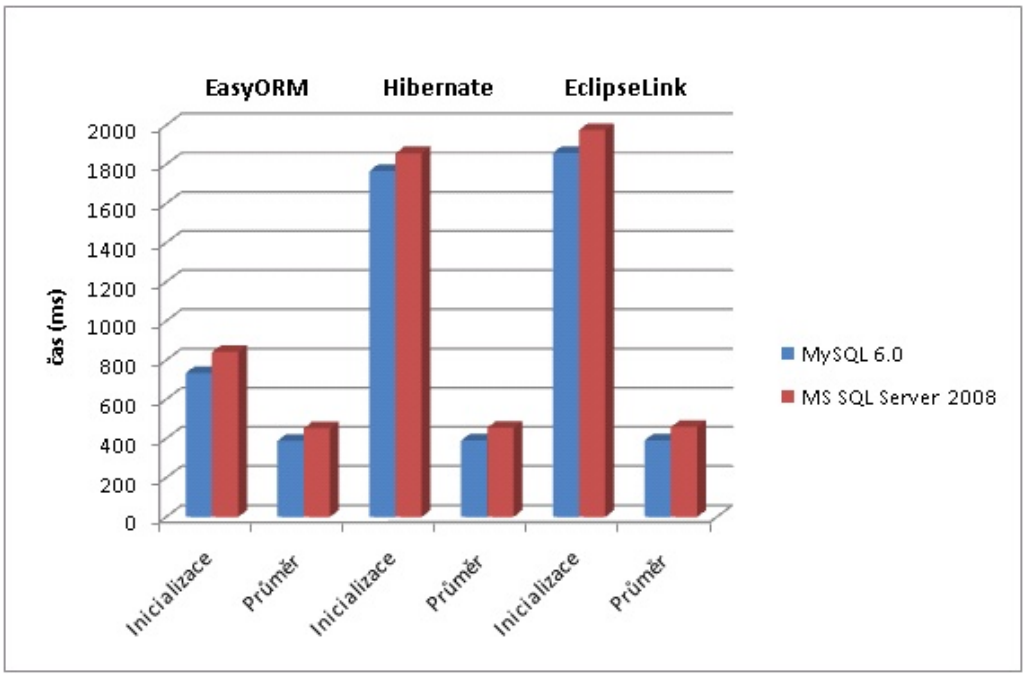
V tabulce 2 jsou zobrazeny časy vykonávání tohoto dotazu u vybraných ORM frameworku. Naměřené hodnoty jsou uváděny v milisekundách. Na obrázku 17 je zobrazen sloupcový graf reprezentující naměřené hodnoty.

Na základě naměřených hodnot bylo určeno toto pořadí frameworků:

1. EasyORM
2. Hibernate
3. EclipseLink

V tomto případě byly výkony jednotlivých frameworků téměř shodné. Výraznější rozdíly byly pouze při inicializaci frameworků. To rozhodlo o celkovém pořadí.

Všimněte si, že dotaz č. 2 vykonal stejnou činnost jako dotaz č. 1 a to v podstatně kratším čase. Potvrdil se tedy předpoklad, že tento postup bude podstatně rychlejší a spotřebuje méně systémových prostředků než u předchozího dotazu.



Obrázek 17: Graf - testovací dotaz č. 2

8.2.3 Dotaz č. 3

8.2.3.1 Implementace Tady jsou zobrazeny ukázky zdrojových kódů, pomocí kterých byl implementován tento dotaz. Jsou zobrazovány pouze určité části kódu, pro přehlednost je vypuštěno například ošetření výjimek.

EasyORM Ve výpisu kódu 26 je zobrazena implementace pro framework EasyORM.

```
 ......
List<Uzivatel> seznam = new ArrayList<Uzivatel>();
PrevodDB prevodDB = new PrevodDB();
List<String> atributy = new ArrayList<String>();
atributy.add("prevod_id");
atributy.add("zdrojovy_ucet");
atributy.add("cilovy_ucet");
atributy.add("castka");
String sql = "FROM Ucet U, Uzivatel Uz, Prevod P WHERE Uz.jmeno = ? AND Uz.prijmeni =
    ?";
sql += "AND Uz.uzivatel_id = U.Uzivatel AND U.ucet_id = P.zdrojovy_ucet";
List<Object> parametry = new ArrayList<Object>();
parametry.add("Petr");
parametry.add("Novák");

seznam = prevodDB.get(atributy, 0, sql, parametry);
 ......
```

Výpis 26: Java – implementace dotazu č. 3 na frameworku EasyORM

EclipseLInk Ve výpisu kódu 27 je zobrazena implementace pro framework EclipseLink.

```
 ......
List<Uzivatel> seznam = new ArrayList<Uzivatel>();
EntityManagerFactory factory = Persistence.createEntityManagerFactory("Tester2PU");
EntityManager em = factory.createEntityManager();

String sql = "SELECT * FROM Ucet U, Uzivatel Uz, Prevod P WHERE Uz.jmeno = #JMENO
    AND Uz.prijmeni = #PRIJMENI ";
sql += "AND Uz.uzivatel_id = U.Uzivatel AND U.ucet_id = P.zdrojovy_ucet";
Query query = em.createNativeQuery(sql);
query.setParameter("JMENO", "Petr");
query.setParameter("PRIJMENI", "Novák");

seznam = query.getResultList();
 ......
```

Výpis 27: Java – implementace dotazu č. 3 na frameworku EclipseLink

Hibernate Ve výpisu kódu 28 je zobrazena implementace pro framework Hibernate.

```
 ......
Session session = null;
session = HibernateUtil.getSessionFactory().openSession();
String sql = "SELECT P FROM Ucet as U, Uzivatel as Uz, Prevod as P WHERE Uz.jmeno =
    :JMENO AND Uz.prijmeni = :PRIJMENI ";
sql += "AND Uz.uzivatelId = U.uzivatel AND U.ucetId = P.ucetByZdrojovyUcet";

Query query = session.createQuery(sql);
query.setString("JMENO", "Petr");
query.setString("PRIJMENI", "Novák");

seznam = query.list();
 ......
```

Výpis 28: Java – implementace dotazu č. 3 na frameworku Hibernate

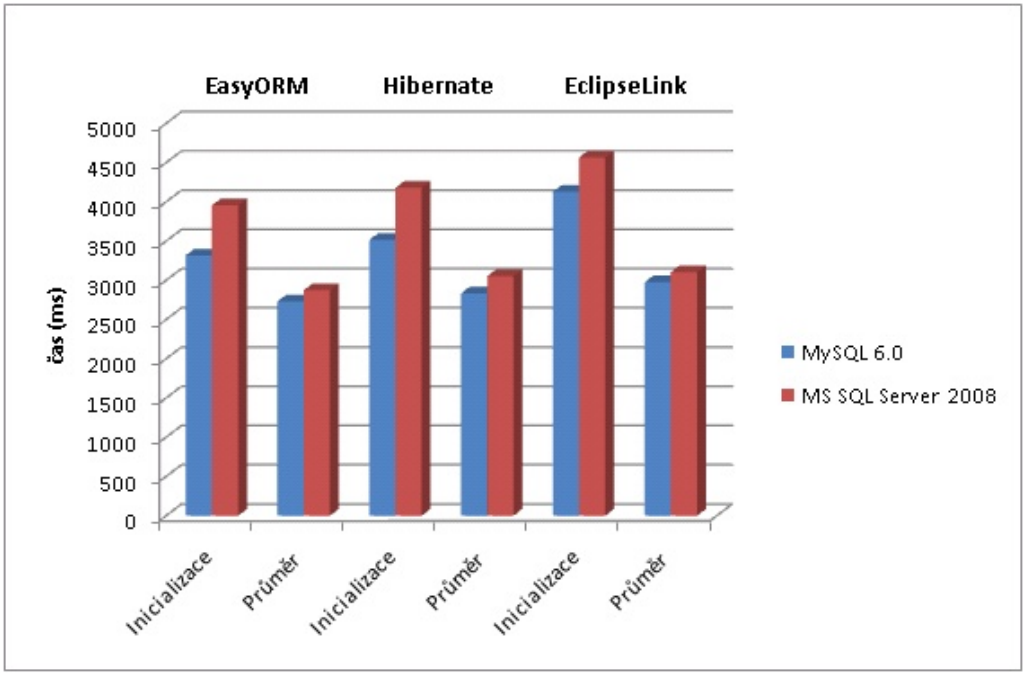
8.2.3.2 Výsledky a vyhodnocení V tabulce 3 jsou zobrazeny časy vykonávání tohoto dotazu u vybraných ORM frameworku. Naměřené hodnoty jsou uváděny v milisekundách. Na obrázku 18 je zobrazen sloupcový graf reprezentující naměřené hodnoty.

Na základě naměřených hodnot bylo určeno toto pořadí frameworků:

1. EasyORM
2. Hibernate

Framework	EasyORM		Hibernate		EclipseLink	
SŘBD	Inicial.	Průměr	Inicial.	Průměr	Inicial.	Průměr
MySQL 6.0	3318	2732	3515	2834	4130	2976
MS SQL Server 2008	3956	2874	4180	3054	4567	3106

Tabulka 3: Testovací dotaz č. 3



Obrázek 18: Graf - testovací dotaz č. 3

3. EclipseLink

V tomto případě byly výkony všech frameworků poměrně shodné. Nejlepší byl framework EasyORM.

Celkové výsledky tohoto dotazu jsou, ale poměrně překvapivé. Bylo předpokládáno, že jeho vykonání bude časově velmi náročné. Rychlost vykonání tohoto dotazu je způsobena velkou operační pamětí a vlastní optimalizací SQL dotazu na straně SŘBD.

8.2.4 Dotaz č. 4

8.2.4.1 Implementace Tady jsou zobrazeny ukázky zdrojových kódů, pomocí kterých byl implementován tento dotaz. Jsou zobrazovány pouze určité části kódu, pro přehlednost je vypuštěno například ošetření výjimek.

EasyORM Ve výpisu kódu 29 je zobrazena implementace pro framework EasyORM.

```
 ......
List<Uzivatel> seznam = new ArrayList<Uzivatel>();
PrevodDB prevodDB = new PrevodDB();
List<String> atributy = new ArrayList<String>();
atributy.add("prevod_id");
atributy.add("zdrojovy_ucet");
atributy.add("cilovy_ucet");
atributy.add("castka");
String sql = "FROM Prevod WHERE zdrojovy_ucet IN (SELECT ucet_id FROM Ucet WHERE
    uzivatel IN (SELECT uzivatel_id FROM Uzivatel WHERE jmeno=? AND prijmeni = ?))";
List<Object> parametry = new ArrayList<Object>();
parametry.add("Petr");
parametry.add("Novák");

seznam = prevodDB.get(atributy, 0, sql, parametry);
 ......
```

Výpis 29: Java – implementace dotazu č. 4 na frameworku EasyORM

EclipseLInk Ve výpisu kódu 30 je zobrazena implementace pro framework EclipseLink.

```
 ......
List<Uzivatel> seznam = new ArrayList<Uzivatel>();
EntityManagerFactory factory = Persistence.createEntityManagerFactory("Tester2PU");
EntityManager em = factory.createEntityManager();

String sql = "SELECT * FROM Prevod WHERE zdrojovy_ucet IN (SELECT ucet_id FROM Ucet
    WHERE uzivatel IN (SELECT uzivatel_id FROM Uzivatel WHERE jmeno = #JMENO AND
    prijmeni = #PRIJMENI))";
Query query = em.createNativeQuery(sql);
query.setParameter("JMENO", "Petr");
query.setParameter("PRIJMENI", "Novák");

seznam = query.getResultList();
```

Framework	EasyORM		Hibernate		EclipseLink	
SŘBD	Inicial.	Průměr	Inicial.	Průměr	Inicial.	Průměr
MySQL 6.0	3140	2687	3420	2758	3937	2453
MS SQL Server 2008	3562	2783	3841	2912	4215	2639

Tabulka 4: Testovací dotaz č. 4

```
.....
```

Výpis 30: Java – implementace dotazu č. 4 na frameworku EclipseLink

Hibernate Ve výpisu kódu 31 je zobrazena implementace pro framework Hibernate.

```
 .....
Session session = null;
session = HibernateUtil.getSessionFactory().openSession();
String sql = "SELECT P FROM Prevod P WHERE ucetByZdrojovyUcet IN (SELECT ucetId
    FROM Ucet WHERE uzivatel IN (SELECT uzivatelId FROM Uzivatel WHERE jmeno = :
    JMENO AND prijmeni = :PRIJMENI))";

Query query = session.createQuery(sql);
query.setString("JMENO", "Petr");
query.setString("PRIJMENI", "Novák");

seznam = query.list();
 .....
```

Výpis 31: Java – implementace dotazu č. 4 na frameworku Hibernate

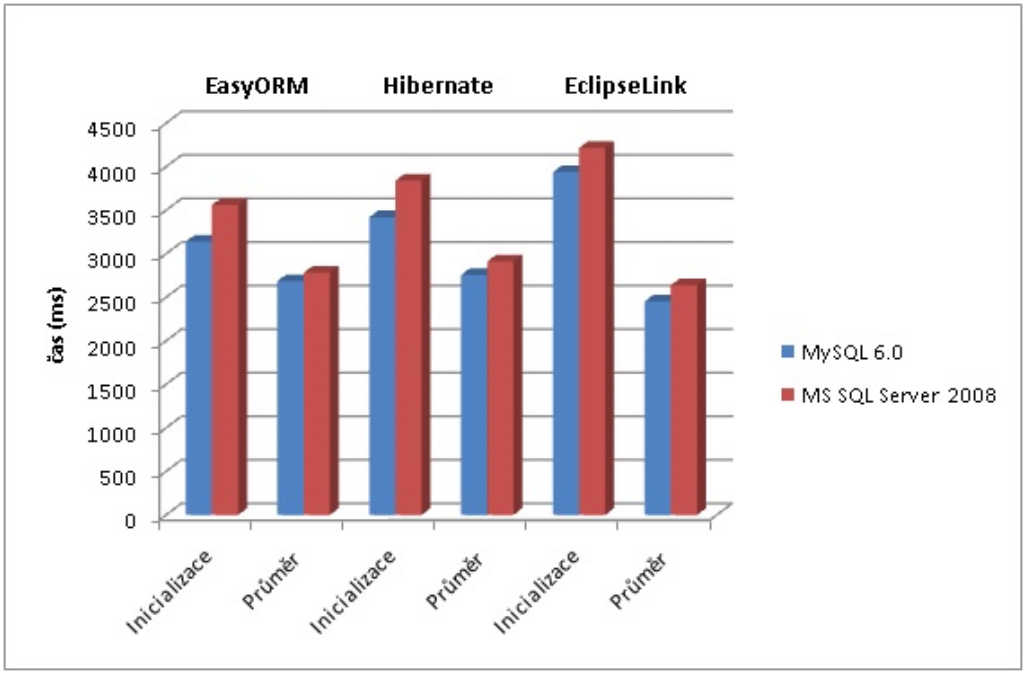
8.2.4.2 Výsledky a vyhodnocení V tabulce 4 jsou zobrazeny časy vykonávání tohoto dotazu u vybraných ORM frameworku. Naměřené hodnoty jsou uváděny v milisekundách. Na obrázku 19 je zobrazen sloupcový graf reprezentující naměřené hodnoty.

Na základě naměřených hodnot bylo určeno toto pořadí frameworků:

1. EclipseLink
2. EasyORM
3. Hibernate

V tomto případě byly výkony frameworků téměř shodné. Framework EclipseLink byl ale nejrychlejší.

Oproti předpokladům je doba vykonávání tohoto dotazu téměř shodná s dotazem č. 3. Je to způsobeno velkou operační pamětí a vlastní optimalizací SQL dotazu na straně SŘBD.



Obrázek 19: Graf - testovací dotaz č. 4

8.2.5 Dotaz č. 5

8.2.5.1 Implementace Tady jsou zobrazeny ukázky zdrojových kódů, pomocí kterých byl implementován tento dotaz. Jsou zobrazovány pouze určité části kódu, pro přehlednost je vypuštěno například ošetření výjimek.

EasyORM Ve výpisu kódu 32 je zobrazena implementace pro framework EasyORM.

```
 ......
Transaction tx = new Transaction();
tx.begin();
for (int i = 0; i<10000; i++)
{
    getUzivatel().save();
}
tx.commit();
 ......
```

Výpis 32: Java – implementace dotazu č. 5 na frameworku EasyORM

EclipseLInk Ve výpisu kódu 33 je zobrazena implementace pro framework EclipseLink.

```
 ......
Uzivatel uzivatel;
EntityManagerFactory factory = Persistence.createEntityManagerFactory("Tester2PU");
```

Framework	EasyORM		Hibernate		EclipseLink	
SŘBD	Inicial.	Průměr	Inicial.	Průměr	Inicial.	Průměr
MySQL 6.0	2281	1719	3766	2375	4344	2863
MS SQL Server 2008	2086	1603	3348	2163	4132	2646

Tabulka 5: Testovací dotaz č. 5

```
EntityManager em = factory.createEntityManager();

EntityTransaction tx = em.getTransaction();
tx.begin();

for (int i = 0; i < 10000; i++)
{
    uzivatel = getUzivatel();
    em.persist(uzivatel);
}
tx.commit() ;.....
```

Výpis 33: Java – implementace dotazu č. 5 na frameworku EclipseLink

Hibernate Ve výpisu kódu 34 je zobrazena implementace pro framework Hibernate.

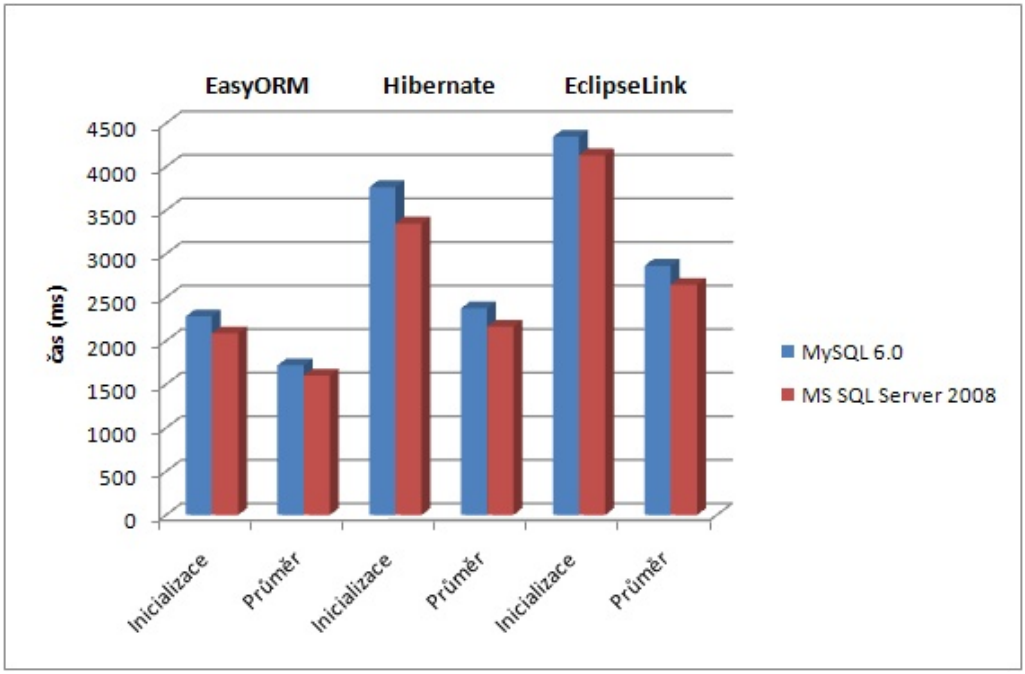
```
 .....
Session session = null;
Transaction tx = null;
Uzivatel uzivatel;

session = HibernateUtil.getSessionFactory().openSession();
tx = session.beginTransaction();

for (int i = 0; i < 10000; i++)
{
    uzivatel = getUzivatel();
    session.save(uzivatel);
}
tx.commit();
 .....
```

Výpis 34: Java – implementace dotazu č. 5 na frameworku Hibernate

8.2.5.2 Výsledky a vyhodnocení V tabulce 5 jsou zobrazeny časy vykonávání tohoto dotazu u vybraných ORM frameworku. Naměřené hodnoty jsou uváděny v milisekundách. Na obrázku 20 je zobrazen sloupcový graf reprezentující naměřené hodnoty.

Na základě naměřených hodnot bylo určeno toto pořadí frameworků:

1. EasyORM
2. Hibernate



Obrázek 20: Graf - testovací dotaz č. 5

3. EclipseLink

Podle očekávání se na prvém místě umístil framewor EasyOrm. Oba další frameworky měli větší režii při vytváření objektového modelu pro 10000 objektů.

8.2.6 Dotaz č. 6

8.2.6.1 Implementace Tady jsou zobrazeny ukázky zdrojových kódů, pomocí kterých byl implementován tento dotaz. Jsou zobrazovány pouze určité části kódu, pro přehlednost je vypuštěno například ošetření výjimek.

EasyORM Ve výpisu kódu 35 je zobrazena implementace pro framework EasyORM.

```
 ......
ExekuceDB exekuceDB = new ExekuceDB();
List<Exekuce> seznam = new ArrayList<Exekuce>();
Transaction tx  = new Transaction();

tx.begin();
seznam = exekuceDB.getAll();

for (Exekuce element : seznam)
{
    element.setStahnout(element.getStahnout()+ 2);
    exekuceDB.update(element);
```

```
}
tx.commit();
 ......
```

Výpis 35: Java – implementace dotazu č. 6 na frameworku EasyORM

EclipseLInk Ve výpisu kódu 36 je zobrazena implementace pro framework EclipseLink.

```
 ......
BigDecimal b = new BigDecimal(500);
List<Exekuce> seznam = new ArrayList<Exekuce>();
EntityManagerFactory factory = Persistence.createEntityManagerFactory("Tester2PU");
EntityManager em = factory.createEntityManager();
EntityTransaction tx = em.getTransaction();

tx.begin();
Query query = em.createQuery("SELECT e FROM Exekuce e");
seznam = query.getResultList();

for (Exekuce element : seznam)
{
    element.setStahnout(element.getStahnout().add(b));
    element.setUzivatel(element.getUzivatel()+1);
    element.setStahnuto(element.getStahnuto().add(b));
}
tx.commit();
 ......
```

Výpis 36: Java – implementace dotazu č. 6 na frameworku EclipseLink

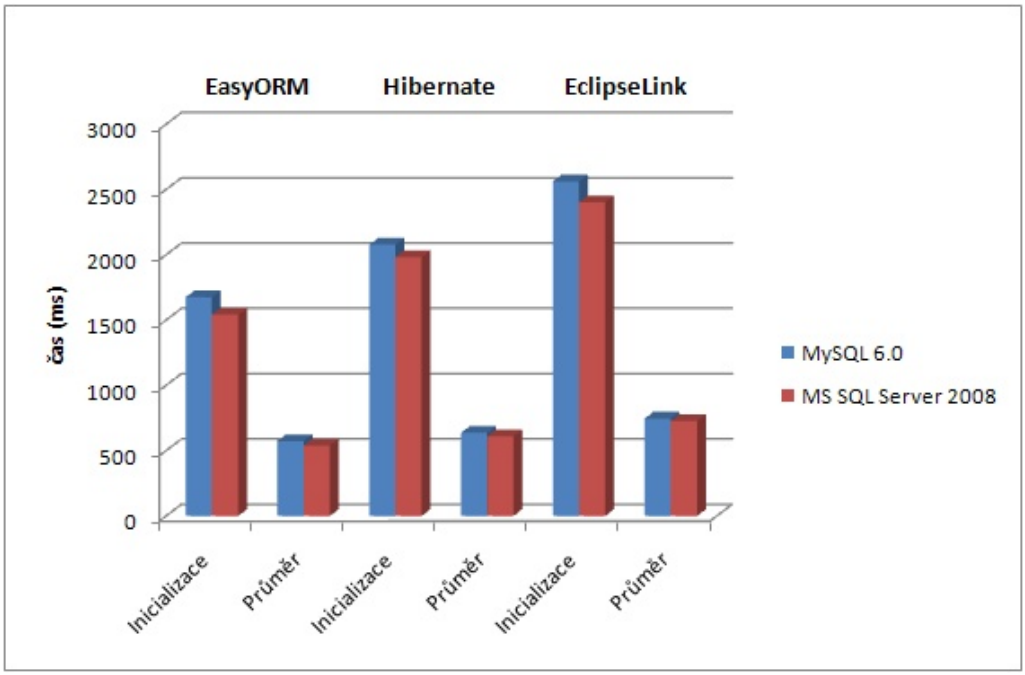
Hibernate Ve výpisu kódu 37 je zobrazena implementace pro framework Hibernate.

```
 ......
Session session = null;
Transaction tx = null;
BigDecimal b = new BigDecimal(500);
List<Exekuce> seznam = new ArrayList<Exekuce>();

session = HibernateUtil.getSessionFactory().openSession();
tx = session.beginTransaction();
seznam = (List<Exekuce>) session.createQuery("from Exekuce").list();

for (Exekuce element : seznam)
{
    element.setStahnout(element.getStahnout().add(b));
    session.save(element);
}
tx.commit();
 ......
```

Výpis 37: Java – implementace dotazu č. 6 na frameworku Hibernate

Framework	EasyORM		Hibernate		EclipseLink	
SŘBD	Inicial.	Průměr	Inicial.	Průměr	Inicial.	Průměr
MySQL 6.0	1675	574	2078	637	2562	746
MS SQL Server 2008	1541	539	1982	609	2403	726

Tabulka 6: Testovací dotaz č. 6



Obrázek 21: Graf - testovací dotaz č. 6

8.2.6.2 Výsledky a vyhodnocení V tabulce 6 jsou zobrazeny časy vykonávání tohoto dotazu u vybraných ORM frameworku. Naměřené hodnoty jsou uváděny v milisekundách. Na obrázku 21 je zobrazen sloupcový graf reprezentující naměřené hodnoty.

Na základě naměřených hodnot bylo určeno toto pořadí frameworků:

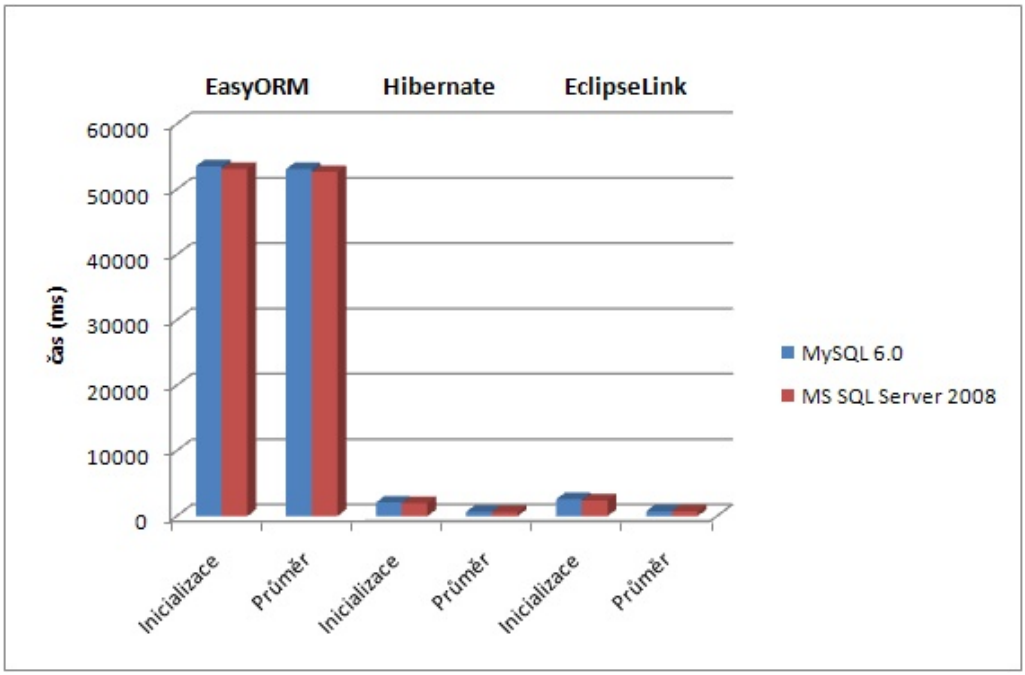
1. EasyORM
2. Hibernate
3. EclipseLink

Na prvním místě se umístil framework EasyORM. Framework EclipseLink automaticky ukládal pouze změněné atributy, proto byly u něj pro lepší porovnání modifikovány všechny atributy.

Na tomto dotazu bude ukázána častá chyba při práci se SŘBD. Touto chybou je nevkládání insertů a updatu do transakce. Podíváme se co se stane, jetliže tento dotaz neuzavřeme do transakce.

Framework	EasyORM		Hibernate		EclipseLink	
SŘBD	Inicial.	Průměr	Inicial.	Průměr	Inicial.	Průměr
MySQL 6.0	53547	53109	2083	641	2573	742
MS SQL Server 2008	53126	52663	1987	605	2407	729

Tabulka 7: Testovací dotaz č. 6B



Obrázek 22: Graf - testovací dotaz č. 6B

8.2.6.3 Výsledky a vyhodnocení V tabulce 7 jsou zobrazeny časy vykonávání tohoto dotazu u vybraných ORM frameworku. Naměřené hodnoty jsou uváděny v milisekundách. Na obrázku 22 je zobrazen sloupcový graf reprezentující naměřené hodnoty.

Na základě naměřených hodnot bylo nyní určeno toto pořadí frameworků:

1. Hibernate
2. EclipseLink
3. EasyORM

V tomto případě framework EasyORM vykonával dotaz neúměrně dlouho. Bylo to proto, že každý jednotlivý update byl vložen do samostatné transakce. Testované SŘBD totiž pracují v takzvaném autocommit módu. V tomto módu vkládají každý update do samostatné transakce, pokud již v některé transakci není. Ostatní frameworky automaticky vložili všechny příkazy update do jediné transakce a tím byly podstatně rychlejší. Proto je velmi důležité vkládat inserty a updaty do transakce.

8.2.7 Dotaz č. 7

8.2.7.1 Implementace Tady jsou zobrazeny ukázky zdrojových kódů, pomocí kterých byl implementován tento dotaz. Jsou zobrazovány pouze určité části kódu, pro přehlednost je vypuštěno například ošetření výjimek.

EasyORM Ve výpisu kódu 38 je zobrazena implementace pro framework EasyORM.

```
 ......
ExekuceDB exekuceDB = new ExekuceDB();
Exekuce exekuce;
List<Exekuce> seznam = new ArrayList<Exekuce>();
Transaction tx  = new Transaction();

tx.begin();
seznam = exekuceDB.getAll();

for (Exekuce element : seznam)
{
    exekuce = new Exekuce();
    exekuce.setStahnout(element.getStahnout()+500);
    exekuce.setExekuce_id(element.getExekuce_id());
    exekuce.setStahnuto(element.getStahnuto());
    exekuce.setUzivatel(element.getUzivatel());
    exekuceDB.update(exekuce, element);
}
tx.commit();
 ......
```

Výpis 38: Java – implementace dotazu č. 7 na frameworku EasyORM

EclipseLInk Ve výpisu kódu 39 je zobrazena implementace pro framework EclipseLink.

```
 ......
BigDecimal b = new BigDecimal(500);
List<Exekuce> seznam = new ArrayList<Exekuce>();
EntityManagerFactory factory = Persistence.createEntityManagerFactory("Tester2PU");
EntityManager em = factory.createEntityManager();
EntityTransaction  tx  = em.getTransaction();

tx.begin();
Query query = em.createQuery("SELECT e FROM Exekuce e");
seznam = query.getResultList();

for (Exekuce element : seznam)
{
    element.setStahnout(element.getStahnout().add(b));
}
tx.commit();
 ......
```

Výpis 39: Java – implementace dotazu č. 7 na frameworku EclipseLink

Framework	EasyORM		Hibernate		EclipseLink	
SŘBD	Inicial.	Průměr	Inicial.	Průměr	Inicial.	Průměr
MySQL 6.0	1125	523	2079	635	1983	598
MS SQL Server 2008	1096	507	1938	599	1876	564

Tabulka 8: Testovací dotaz č. 7

8.2.7.1.1 Hibernate Ve výpisu kódu 40 je zobrazena implementace pro framework Hibernate.

```
 ......
Session session = null;
Transaction tx = null;
BigDecimal b = new BigDecimal(500);
List<Exekuce> seznam = new ArrayList<Exekuce>();

session = HibernateUtil.getSessionFactory().openSession();
tx = session.beginTransaction();
seznam = (List<Exekuce>) session.createQuery("from Exekuce").list();

for (Exekuce element : seznam)
{
   element.setStahnout(element.getStahnout().add(b));
   session.save(element);
}
tx.commit();
 ......
```

Výpis 40: Java – implementace dotazu č. 7 na frameworku Hibernate

8.2.7.2 Výsledky a vyhodnocení V tabulce 8 jsou zobrazeny časy vykonávání tohoto dotazu u vybraných ORM frameworku. Naměřené hodnoty jsou uváděny v milisekundách. Na obrázku 23 je zobrazen sloupcový graf reprezentující naměřené hodnoty.

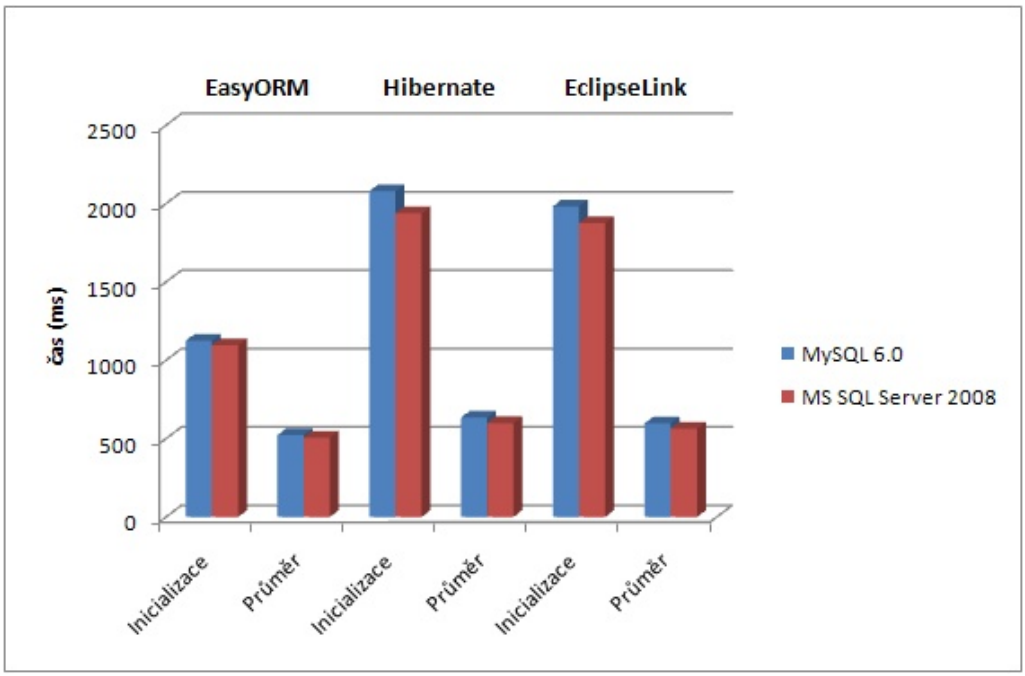
Na základě naměřených hodnot bylo určeno toto pořadí frameworků:

1. EasyORM
2. EclipseLink
3. Hibernate

V tomto případě se nejlépe umístil framework EasyORM. Druhý skončil framework EclipseLink. Přestože byl změněný pouze jeden atribut, framework Hibernate ukládal hodnoty všech atributů. To se projevilo na jeho výkonu.

8.2.8 Dotaz č. 8

8.2.8.1 Implementace Tady jsou zobrazeny ukázky zdrojových kódů, pomocí kterých byl implementován tento dotaz. Jsou zobrazovány pouze určité části kódu, pro přehlednost je vypuštěno například ošetření výjimek.



Obrázek 23: Graf - testovací dotaz č. 7

EasyORM Ve výpisu kódu 41 je zobrazena implementace pro framework EasyORM.

```
 ......
DebetDB debetDB = new DebetDB();
Debet debet;
List<Debet> seznam = new ArrayList<Debet>();

Transaction tx  = new Transaction();
tx.begin();

seznam = debetDB.getAll();
for  (Debet element : seznam)
{
    debet = new Debet();
    debet.setPovoleno(element.getPovoleno() + 10000);
    debet.setCerpano(element.getCerpano());
    debet.setPovoleno(element.getPovoleno());
    debet.setDebet_id(element.getDebet_id());
    debetDB.update(debet, element);
}

tx.commit();
 ......
```

Výpis 41: Java – implementace dotazu č. 8 na frameworku EasyORM

EclipseLInk Ve výpisu kódu 42 je zobrazena implementace pro framework EclipseLink.

```
 ......
BigDecimal b = new BigDecimal(10000);
List<Debet> seznam = new ArrayList<Debet>();
EntityManagerFactory factory = Persistence.createEntityManagerFactory("Tester2PU");
EntityManager em = factory.createEntityManager();

EntityTransaction tx = em.getTransaction();
tx.begin();
Query query = em.createQuery("SELECT d FROM Debet d");
seznam = query.getResultList();

for (Debet element : seznam)
{
    element.setPovoleno(element.getPovoleno().add(b));
}
tx.commit();
 ......
```

Výpis 42: Java – implementace dotazu č. 8 na frameworku EclipseLink

8.2.8.1.1 Hibernate Ve výpisu kódu 43 je zobrazena implementace pro framework Hibernate.

```
 ......
Session session = null;
Transaction tx = null;
BigDecimal b = new BigDecimal(10000);

session = HibernateUtil.getSessionFactory().openSession();
tx = session.beginTransaction();
List<Debet> seznam = new ArrayList<Debet>();
seznam = (List<Debet>) session.createQuery("from Debet").list();

for (Debet element : seznam)
{
    element.setPovoleno(element.getPovoleno().add(b));
}
tx.commit();
 ......
```

Výpis 43: Java – implementace dotazu č. 8 na frameworku Hibernate

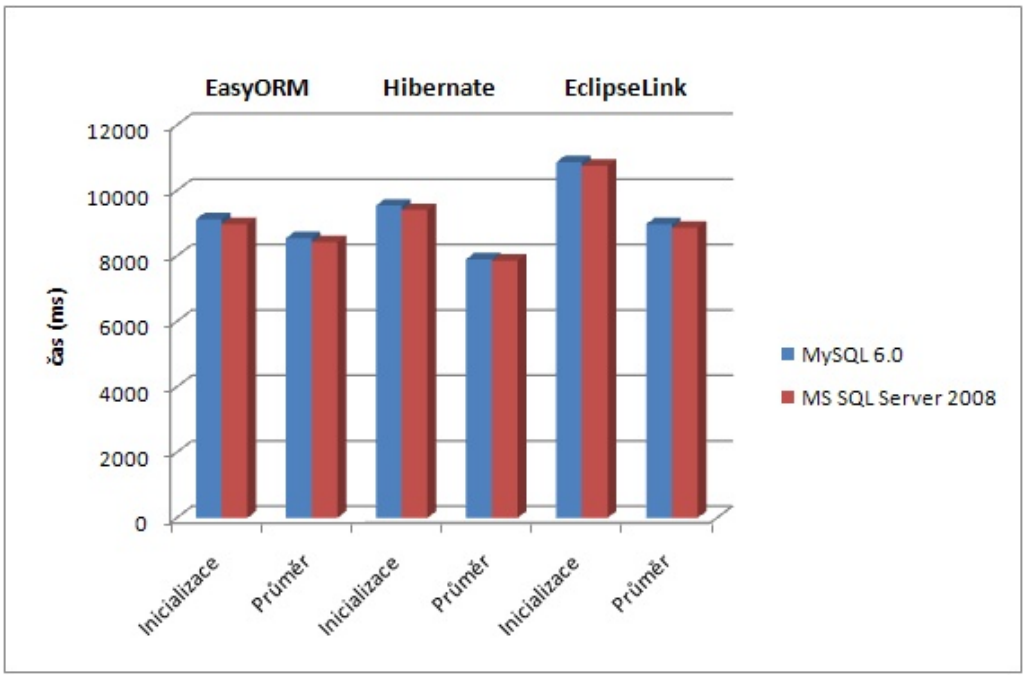
8.2.8.2 Výsledky a vyhodnocení V tabulce 9 jsou zobrazeny časy vykonávání tohoto dotazu u vybraných ORM frameworku. Naměřené hodnoty jsou uváděny v milisekundách. Na obrázku 24 je zobrazen sloupcový graf reprezentující naměřené hodnoty.

Na základě naměřených hodnot bylo určeno toto pořadí frameworků:

1. Hibernate

Framework	EasyORM		Hibernate		EclipseLink	
SŘBD	Inicial.	Průměr	Inicial.	Průměr	Inicial.	Průměr
MySQL 6.0	9126	8548	9543	7906	10869	8978
MS SQL Server 2008	8972	8422	9407	7861	10756	8867

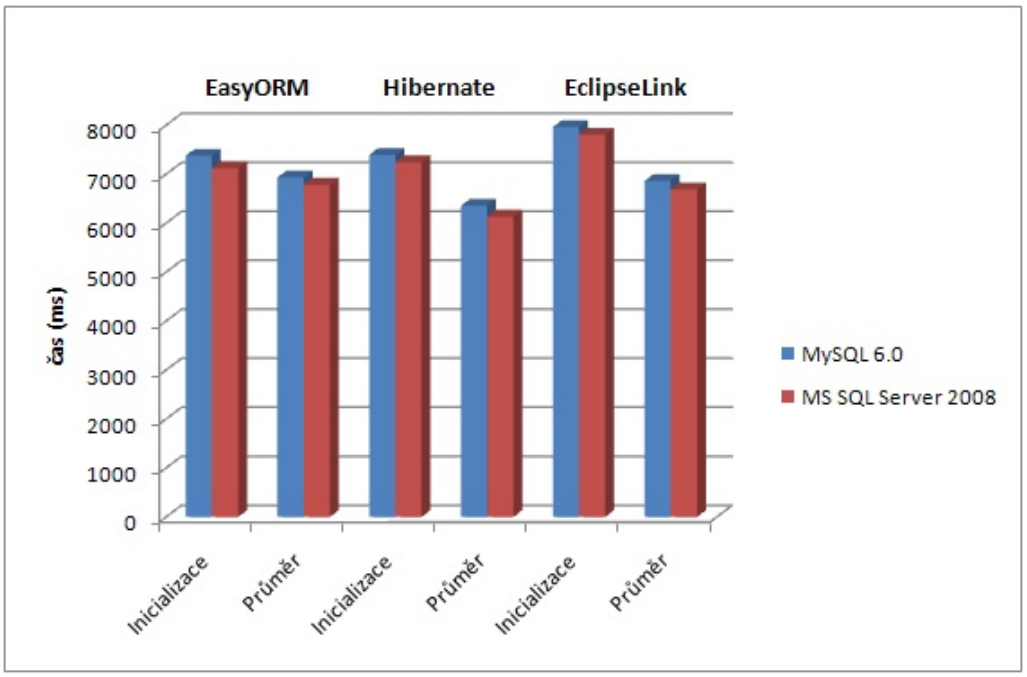
Tabulka 9: Testovací dotaz č. 8



Obrázek 24: Graf - testovací dotaz č. 8

Framework	EasyORM		Hibernate		EclipseLink	
SŘBD	Inicial.	Průměr	Inicial.	Průměr	Inicial.	Průměr
MySQL 6.0	7380	6935	7397	6359	7963	6863
MS SQL Server 2008	7126	6786	7245	6135	7813	6691

Tabulka 10: Testovací dotaz č. 8B



Obrázek 25: Graf - testovací dotaz č. 8B

2. EasyORM
3. EclipseLink

Při úrovni izolace transakcí SERIALIZABLE, se nejlépe umístil framework Hibernate. Teď se podíváme na vyhodnocení dotazu při nastavení úrovně izolace transakci na READ_COMMITED.

8.2.8.3 Výsledky a vyhodnocení V tabulce 10 jsou zobrazeny časy vykonávání tohoto dotazu u vybraných ORM frameworku. Naměřené hodnoty jsou uváděny v milisekundách. Na obrázku 25 je zobrazen sloupcový graf reprezentující naměřené hodnoty.

Na základě naměřených hodnot bylo určeno toto pořadí frameworků:

1. Hibernate
2. EclipseLink

3. EasyORM

I v tomto případě se nejlépe umístil framework Hibernate. To ale nebylo primárním cílem tohoto dotazu. Testovali jsme hlavě to, jaký bude výkon při různých úrovních izolace. Při úrovni READ_COMMITED je výkon vyšší, než při úrovni SERIALIZABLE.

8.2.9 Dotaz č. 9

8.2.9.1 Implementace Tady jsou zobrazeny ukázky zdrojových kódů, pomocí kterých byl implementován tento dotaz. Jsou zobrazovány pouze určité části kódu, pro přehlednost je vypuštěno například ošetření výjimek.

EasyORM Ve výpisu kódu 44 je zobrazena implementace pro framework EasyORM.

```
 ......
ZustatekDB zustatekDB = new ZustatekDB();
 List<Zustatek> seznam = new ArrayList<Zustatek>();

seznam = zustatekDB.getAll(10);

for (Zustatek element : seznam)
{
    soucet += element.getCastka();
}
 ......
```

Výpis 44: Java – implementace dotazu č. 9 na frameworku EasyORM

EclipseLInk Ve výpisu kódu 45 je zobrazena implementace pro framework EclipseLink.

```
 ......
 List<Zustatek> seznam = new ArrayList<Zustatek>();
EntityManagerFactory factory = Persistence.createEntityManagerFactory("Tester2PU");
EntityManager em = factory.createEntityManager();

em = factory.createEntityManager();
Query query = em.createQuery("SELECT z FROM Zustatek z");
seznam = query.getResultList();

for (Zustatek element : seznam)
{
    soucet += element.getCastka().doubleValue();
}
 ......
```

Výpis 45: Java – implementace dotazu č. 9 na frameworku EclipseLink

Framework	EasyORM		Hibernate		EclipseLink	
SŘBD	Inicial.	Průměr	Inicial.	Průměr	Inicial.	Průměr
MySQL 6.0	1043	489	3547	2162	5735	3126
MS SQL Server 2008	1215	523	3702	2275	5834	3369

Tabulka 11: Testovací dotaz č. 9

Hibernate Ve výpisu kódu 46 je zobrazena implementace pro framework Hibernate.

```
 ......
Session session = null;
Transaction tx = null;
BigDecimal soucet = null;

session = HibernateUtil.getSessionFactory().openSession();

List<Zustatek> seznam = new ArrayList<Zustatek>();
seznam = (List<Zustatek>) session.createQuery("from Zustatek").list();

for (Zustatek element : seznam)
{
   soucet.add(element.getCastka()) ;
}
 ......
```

Výpis 46: Java – implementace dotazu č. 9 na frameworku Hibernate

8.2.9.2 Výsledky a vyhodnocení V tabulce 11 jsou zobrazeny časy vykonávání tohoto dotazu u vybraných ORM frameworku. Naměřené hodnoty jsou uváděny v milisekundách. Na obrázku 26 je zobrazen sloupcový graf reprezentující naměřené hodnoty.

Na základě naměřených hodnot bylo určeno toto pořadí frameworků:

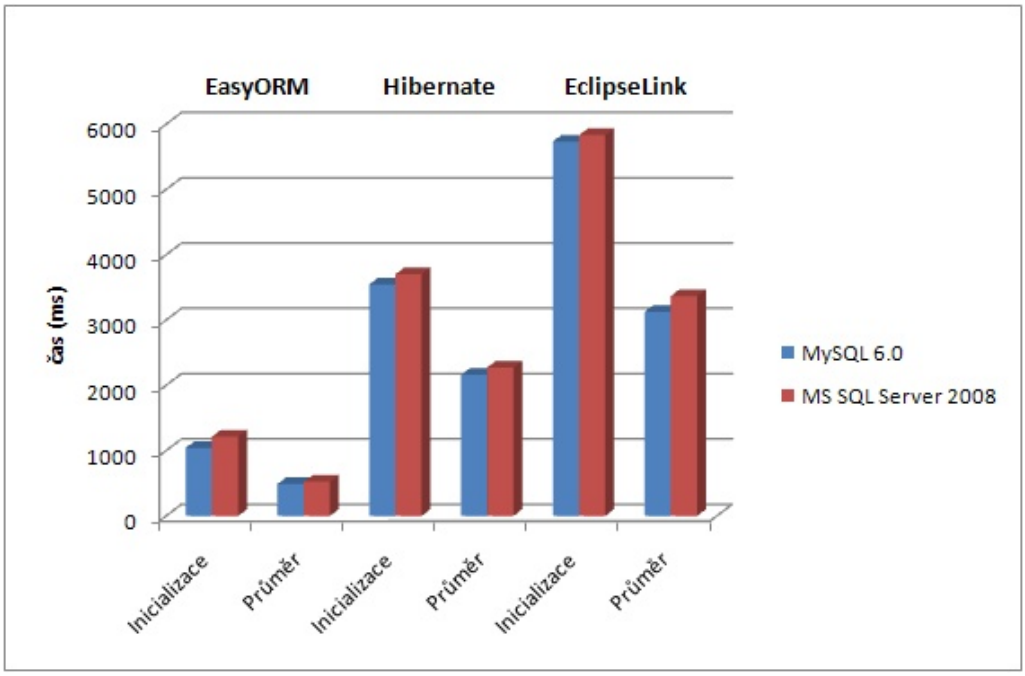
1. EasyORM
2. Hibernate
3. EclipseLink

Na prvním místě se umístil framework EasyORM. V tomto případě ještě nebylo využito cachování frameworků. Nyní byl stejný dotaz spuštěn ve dvou vláknech.

8.2.9.3 Výsledky a vyhodnocení V tabulce 12 jsou zobrazeny časy vykonávání tohoto dotazu u vybraných ORM frameworku. Naměřené hodnoty jsou uváděny v milisekundách. Na obrázku 27 je zobrazen sloupcový graf reprezentující naměřené hodnoty.

Na základě naměřených hodnot bylo nyní určeno toto pořadí frameworků:

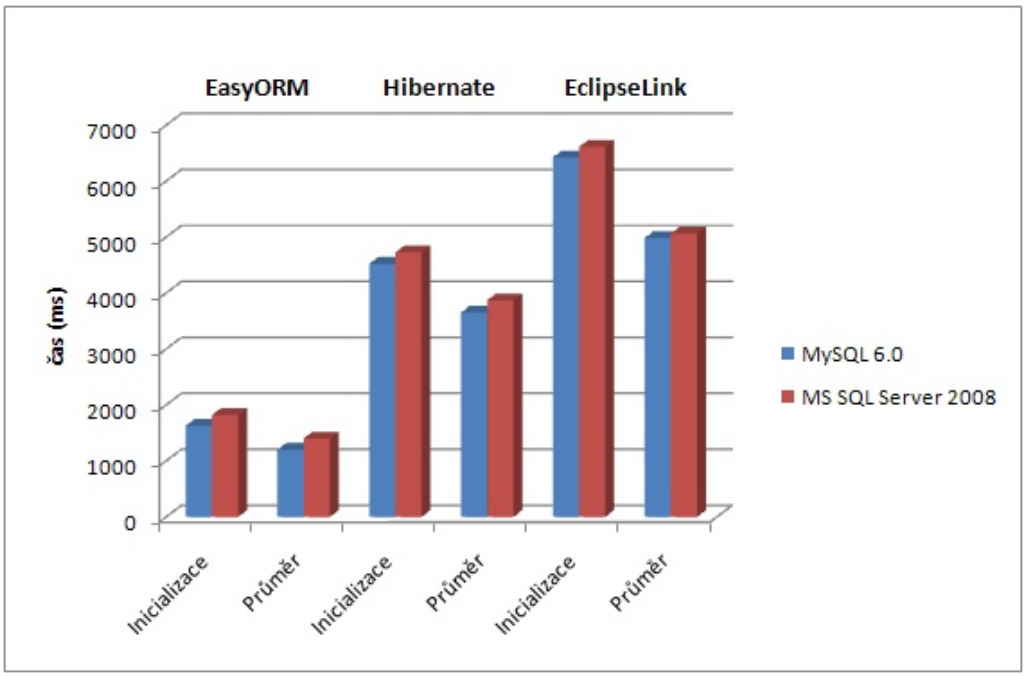
1. EasyORM



Obrázek 26: Graf - testovací dotaz č. 9

Framework	EasyORM		Hibernate		EclipseLink	
SŘBD	Inicial.	Průměr	Inicial.	Průměr	Inicial.	Průměr
MySQL 6.0	1631	1203	4526	3656	6427	4984
MS SQL Server 2008	1823	1403	4729	3874	6613	5067

Tabulka 12: Testovací dotaz č. 9B



Obrázek 27: Graf - testovací dotaz č. 9B

2. Hibernate
3. EclipseLink

V tomto případě byly spuštěny dva dotazy současně. Teoreticky by doba vykonání těchto dotazů, měla být dvakrát tak dlouhá. Na výsledcích je vidět více než dvojnásobný nárůst doby u frameworku EasyORM. Je to způsobeno tím, že tento framework nemá implementovánu podporu cachování a určitou režii má i spravá vláken. U frameworků Hibernate a EclipseLink, je naopak vidět značná úspora času, díky využití cachování. Jak je ale ve výsledcích vidět, pořadí frameworků se nijak nezměnilo. Úspora získaná díky cachování, nevyrovnala výhodu absence objektového modelu u frameworku EasyORM.

8.3 Vyhodnocení porovnávání

Z výsledku porovnávání vyplývá, že implemenovaný framework EasyORM splňuje všechny požadavky, které byly při jeho návrhu specifikovány. Jeho velkou výhodou je absence objektového modelu, která mu umožňuje dosáhnou vyšší výkonnost. Porovnáním se dvěma existujícími implementacemi objektově relačního mapování, byla ověřena kvalita jeho návrhu a implementace. Ve všech porovnávaných oblastech byl plně srovnatelný s ostatními frameworky. V prvním dotazu, který pracoval s miliónem záznamů, byl dokonce výrazně lepší. Celkem v sedmi dotazech z devíti se umístil na prvním místě. Jeho silnou stránkou je práce s velkým počtem záznamů, kde se významně projevuje výhoda absence objektového modelu.

9 Závěr

Tato práce se zabývala objektově-relačním mapováním na platformě Java. Byly v ní prezentovány různé způsoby a možnosti zajištění perzistence dat. Seznámila také čtenáře s vývojem databází a popisovala výhody různých typů používaných SŘBD. Probírala různé techniky objektově relačního mapování a ukázala standardy používané v této oblasti. Představila ORM frameworky pro platformu Java popsala různé techniky pro dosažení vyššího výkonu ORM.

Cílem této práce bylo implementovat některou z technik objektově relačního mapování. Výsledkem této práce je framework EasyORM, který splňuje všechny požadavky, které byly při jeho návrhu specifikovány. Porovnáním se dvěma existujícími implementacemi objektově relačního mapování byla ověřena kvalita jeho návrhu a implementace. Ve všech oblastech byl s porovnávanými frameworky minimálně srovnatelný. V jedné z nich je dokonce výrazně předčil. Jeho silnou stránkou je práce s velkým počtem záznamů, kde se významně projevuje výhoda absence objektového modelu. Pokud bude implementován v datové vrstvě informačního systému, může výrazným způsobem přispět ke stabilitě a zvýšení výkonu tohoto systému.

Framework byl navrhován pro využití v datové vrstvě informačního systému, ale může být použitý v jakékoliv aplikací pro platformu Java. Může rovněž používat jakýkoliv SŘBD, ke kterému je dostupný JDBC konektor.

Jako hlavní možnost dalšího vývoje tohoto framewworku, se jeví implementace podpory hromadných operací. Každý SŘBD má totiž své specifické vlastnosti a metody, které umožňují, dosáhnou nejlepšího výkonu tohoto databázového systému. Pokud bude framework EasyORM vyladěn na provádění hromadných operací pro konkrétní SŘBD, bude na něm tyto operace provádět mnohem rychleji. Čímž by se významně urychlilo zpracování velkého množství záznamů.

V této práci se už z časových důvodů nestihla tato podpora implementovat. Pokud by byla implementována pro nejčastěji používané SŘBD, byla by užitná hodnota tohoto frameworku větší.

V závěru je důležité zdůraznit toto. ORM nástroje a frameworky sice mohou výrazným způsobem urychlit vývoj informačního systému, ale je třeba jim rozumět a umět je správně použít. Jejich nesprávným použitím muže dojít k výraznému snížení výkonu vytvářeného informačního systému. Není možné je začít efektivně používat, bez dobré znalosti technik ORM a SŘBD.

10 Literatura

[1] GABHART, Kyle. *J2EE pathfinder: Persistent data management* [online]. c2003, poslední revize 2.4.2003 [cit.2010-02-05]. Dostupné z: <http://www.ibm.com/developerworks/java/library/j-pj2ee3.html>.

[2] *JDK 6 Serialization* [online]. c2005 [cit.2010-04-15]. Dostupné z: <http://java.sun.com/javase/6/docs/technotes/guides/serialization/>.

[3] *Java Platform SE 6* [online]. c2009, poslední revize 18.12.2009 [cit.2010-02-05]. Dostupné z: <http://java.sun.com/javase/6/docs/api/>.

[4] *Extensible Markup Language (XML)* [online]. c2003, poslední revize 13.3.2010 [cit.2010-02-05]. Dostupné z: <http://www.w3.org/XML/>.

[5] Žák, Karel. *Historie relačních databází* [online]. c2001, poslední revize 19.10.2001 [cit.2010-02-05]. Dostupné z: <http://www.root.cz/clanky/historie-relacnich-databazi/>.

[6] Hector Garcia-Molina, Jennifer Widom, Jeffrey D. Ullman *Database Systems: The Complete Book* 2. vyd. Prentice Hall, 2008. 1248 s. ISBN 978-0131873254

[7] Date C.J. *An Introduction to Database Systems* 8. vyd. Addison Wesley, 2003. 1024 s. ISBN 978-0321197849

[8] *ODBMS.ORG :: Object Database (ODBMS) | Object-Oriented Database (OODBMS) | Free Resource Portal* [online]. c2010, [cit.2010-04-05]. Dostupné z: <http://www.odbms.org/Introduction/definition.aspx>.

[9] Douglas K. Barry *The Object Database Handbook* 1. vyd. Wiley, 1996. 352 s. ISBN 978-0471147183

[10] Stonebraker Michael *Object-Relational DBMSs: The Next Great Wave* 2. vyd. Morgan Kaufmann Pub, 1996. 216 s. ISBN 978-1558603974

[11] *Cache - Wikipedia, the free encyclopedia* [online]. 18.10.2001, poslední revize 21.4.2010 [cit.2010-04-22]. Dostupné z: <http://en.wikipedia.org/wiki/Cache>.

[12] *InterSystems Caché - Otázky a odpovědi* [online]. c2010, [cit.2010-04-05]. Dostupné z: <http://www.intersystems.cz/cache/cache-qa.htm>.

[13] Lambert M. Surhone *Object-Relational Mapping* 1. vyd. Betascript Publishing, 2009. 104 s. ISBN 978-6130509668

[14] *Java Data Objects (JDO)* [online]. 25.9.2007, poslední revize 25.9.2007 [cit.2010-04-24]. Dostupné z: <http://java.sun.com/jdo/>.

[15] Keith Mike, Schincariol Merrick *Pro JPA 2: Mastering the Java Persistence API* 1. vyd. Apress, 2009. 500 s. ISBN 978-1430219569

[16] *The Java Persistence API - A Simpler Programming Model for Entity Persistence* [online]. 4.5.2006, poslední revize 4.5.2006 [cit.2010-04-24]. Dostupné z: <http://java.sun.com/developer/technicalArticles/J2EE/jpa/>.

[17] KING Gavin, BAUER Christian, ANDERSEN Max Rydahl, BERNARD Emmanuel and EBERSOLE Steve *HIBERNATE - Relational Persistence for Idiomatic Java* [online]. c2004, poslední revize 14.4.2010 [cit.2010-04-24]. Dostupné z: <http://docs.jboss.org/hibernate/stable/core/reference/en/html/>.

[18] BAUER Christian and KING Gavin. *Hibernate in Action.* 1. vyd. Greenwich: Manning Publications Co., 2004. 408 s. ISBN 1932394-15-X

[19] *EclipseLink - Eclipsepedia* [online]. poslední revize 13.4.2010 [cit.2010-04-24]. Dostupné z: <http://wiki.eclipse.org/EclipseLink/Development/Architecture/EclipseLink>.

[20] *EclipseLink/Development/Architecture/EclipseLink - Eclipsepedia* [online]. poslední revize 8.12.2008 [cit.2010-04-24]. Dostupné z: <http://wiki.eclipse.org/EclipseLink/Development/Architecture/EclipseLink>.

[21] *Oracle TopLink* [online]. [cit.2010-04-24]. Dostupné z: <http://www.oracle.com/technology/products/ias/toplink/index.html>.

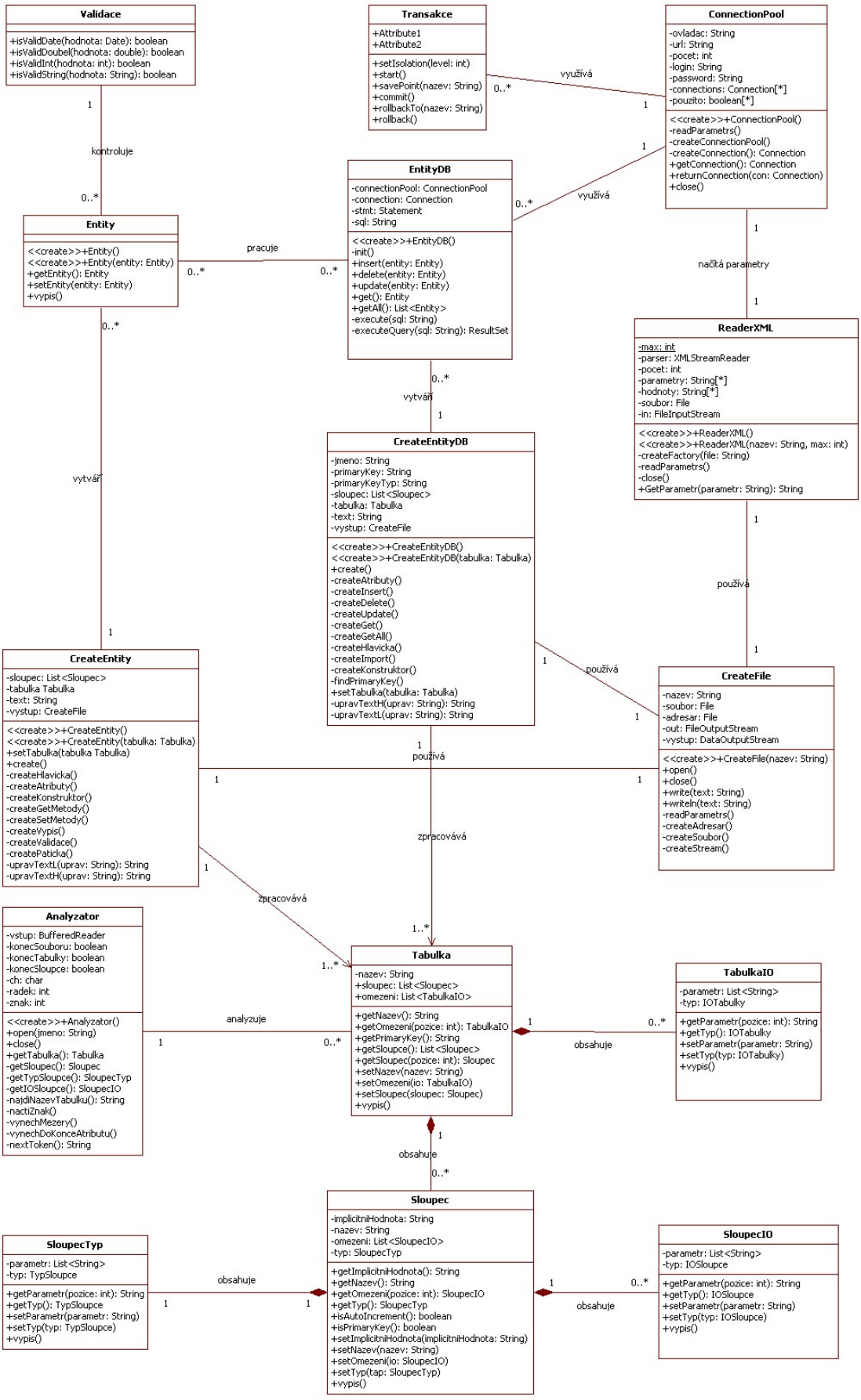
[22] Krátký Michal *Tvorba informačních systémů* [pdf dokument]. 10.4.2010, poslední revize 31.3.2010 [cit.2010-04-24].

[23] *Mapping Objects to Relational Databases: O/R Mapping In Detail* [online]. [cit.2010-04-24]. Dostupné z: <http://www.agiledata.org/essays/mappingObjects.html>.

[24] *Developer Resources for Java Technology* [online]. [cit.2010-04-24]. Dostupné z: <http://java.sun.com/>.

[25] *Vítejte u NetBeans* [online]. [cit.2010-04-24]. Dostupné z: <http://netbeans.org/index_cs.html>.

A Diagram tříd



Obrázek 28: Diagram tříd

B Obsah přiloženého CD

Součástí této práce je také CD, které obsahuje elektronickou podobu tohoto textu ve formátu pdf. Také obsahuje programátorskou dokumentaci a implementaci frameworku EasyORM. Jsou na něm rovněž volně šiřitelné knihovny a nástroje.Adresářová struktura je následující:

- /text – elektronická podoba textu práce
- /doc – programátorská dokumentace
- /source – implementace framworku EasyORM
- /tools – volně šiřitelné knihovny a nástroje

C Použití frameworku

Podrobné informace o použití frameworku jsou uvedeny v dokumentaci. Generování tříd spustíte pomocí příkazu: java -jar „easyORM.jar“ „název skriptu“